Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1997-03

Model management via dependencies between variables: an indexical reasoning in mathematical modeling

Rehber, Devrim

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/26086

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

NPS ARCHIVE
1997.03
REHBER, D.

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

MODEL MANAGEMENT VIA DEPENDENCIES BETWEEN VARIABLES: AN INDEXICAL REASONING IN MATHEMATICAL MODELING

by

Devrim Rehber

March, 1997

Principal Advisor: Hemant K. Bhargava
Associate Advisor: Gordon H. Bradley

Approved for public release; distribution is unlimited.

Thesis
R322

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE
MODEL MANAGEMENT VIA DEPENDENCIES BETWEEN VARIABLES: AN INDEXICAL REASONING IN MATHEMATICAL MODELING

5. FUNDING NUMBERS

6. AUTHOR(S) Rehber, Devrim

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING /MONITORING ORGANIZATION REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The design and implementation of computer-based modeling systems and environments are gaining interest and importance in decision sciences and information systems. In spite of the increasing popularity of GUI-based operating systems, most of the algebraic modeling languages, today, are still file-oriented, text-based, and therefore require structured declarations and formal model definitions. The utilization of the standard graphical screen objects of a graphics-based operating system provides enhanced visualization of models and more cohesive human-computer interaction.

The approach taken in this thesis is to explore the design and implementation of a graph-based modeling system focusing on computational dependencies between model components. Another important aspect of this research is the development of a user-friendly model formulation interface for algebraic modeling languages and systems; these facilitate the description and implementation of mathematical models by allowing the modeler to employ commonly known and powerful algebraic notation instead of language specific codes.

The major conclusion of this thesis is that dependencies between variables are a foundation for building and using models and modeling languages. It also shows that this supports model documentation, validation, formulation, implementation, comprehension, maintenance and reuse. That is, it impacts nearly every step of the modeling life cycle.

14. SUBJECT TERMS Model Management, Decision Support Systems, Dependency Diagrams, Modeling Languages, Mathematical Modeling, Graph-based Modeling, Decision Science.			15. NUMBER OF PAGES 70
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

MODEL MANAGEMENT VIA DEPENDENCIES BETWEEN VARIABLES: AN INDEXICAL REASONING IN MATHEMATICAL MODELING

Devrim Rehber
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1990

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE
IN
INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
March 1997

NPS ARCHIVE
1997.03
REHBER, D.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

ABSTRACT

The design and implementation of computer-based modeling systems and environments are gaining interest and importance in decision sciences and information systems. In spite of the increasing popularity of GUI-based operating systems, most of the algebraic modeling languages, today, are still file-oriented, text-based, and therefore require structured declarations and formal model definitions. The utilization of the standard graphical screen objects of a graphics-based operating system provides enhanced visualization of models and more cohesive human-computer interaction.

The approach taken in this thesis is to explore the design and implementation of a graph-based modeling system focusing on computational dependencies between model components. Another important aspect of this research is the development of a user-friendly model formulation interface for algebraic modeling languages and systems; these facilitate the description and implementation of mathematical models by allowing the modeler to employ commonly known and powerful algebraic notation instead of language specific codes.

The major conclusion of this thesis is that dependencies between variables are a foundation for building and using models and modeling languages. It also shows that this supports model documentation, validation, formulation, implementation, comprehension, maintenance and reuse. That is, it impacts nearly every step of the modeling life cycle.

TABLE OF CONTENTS

I. INTRODUCTION 1

A. OBJECTIVES 1

B. BACKGROUND 1

1. Model Abstraction 1

2. Decision Support Systems 2

3. Data 4

4. Graphics 4

5. Structured Modeling 4

6. Modeling Languages 5

7. Summary 6

C. RESEARCH QUESTIONS 6

D. METHODOLOGY 7

E. CHAPTER OUTLINE 7

II. CONCEPTUAL FRAMEWORK 9

A. OVERVIEW 9

1. Model Variables and Dependencies 9

a. Dependency Diagrams 10

b. Dependency Types 11

c. User Interface 12

d. Model Manager 12

2. Dependencies in Modeling and the Modeling Life Cycle 13

a. Dependency Diagram: EOQ Example 13

b. Dependencies in the EOQ Model 14

c. Generation of Model Expressions (Functional Form) 15

d. Dependency Diagrams: Transportation Model 16

e. Dependencies in the Transportation Model 17

f. Generation of Model Expressions (Functional Form) 17

g. Generation of Equational Forms Using Indexical Rules 18

3. Dependencies in Other Domains 19

B. SUMMARY 20

III. REASONING WITH DEPENDENCIES 21

A. OVERVIEW 21

B. FRAMEWORK 21

1. Objects, Terminology, and Notation 21

a. Example 1 (Assignment Constraint) 22

b. Example 2 (Subtour Elimination Constraint) 22

2. Indexical Reasoning 23

3. Dimensional Balance 24

a. Laws of Dimensional Consistency 25

b. Laws of Dimensional Arithmetic 26

4. Model Type 26

IV. IMPLEMENTATION 29

A. APPLICATION DEVELOPMENT TOOL – BORLAND'S DELPHI™ 29

B. APPLICATION IMPLEMENTATION 29

1. Functional Description of the Program 29

2. Program Interface 29

C. MAINTENANCE 42

V. CONCLUSIONS 43

A. SUMMARY 43

B. CONTRIBUTIONS 44

C. AREAS OF FURTHER RESEARCH 45

LIST OF REFERENCES 47

INITIAL DISTRIBUTION LIST 51

LIST OF FIGURES

Figure 1. Levels of Abstraction 2
Figure 2. DSS Components 3
Figure 3. Elements of a Dependency Diagram 10
Figure 4. Dependencies 10
Figure 5. Dependency Diagram for the EOQ Model 14
Figure 6. About Screen of Model Manager 30
Figure 7. Welcome Page 31
Figure 8. Model Information Screen 31
Figure 9. Dependencies Page 32
Figure 10. Input Box for Set Properties 33
Figure 11. Input Box for Variable Properties 33
Figure 12. Input Box for Dependency Declaration 34
Figure 13. Right-Click on a Component 35
Figure 14. Properties Message Box 35
Figure 15. Delete Confirmation Message 36
Figure 16. Right-Click on an Empty Area 36
Figure 17. Message Box for Listing the Sets 37
Figure 18. Message Box for Listing the Variables 37
Figure 19. Message Box for Listing the Dependencies 38
Figure 20. Functions Page 39
Figure 21. Equations Page 40
Figure 22. Exporter Page 41
Figure 23. Exit Confirmation Dialog Box 42

LIST OF TABLES

Table I. Objects and Notations 23

ACKNOWLEDGMENT

The author wants to thank Prof. Hemant K. Bhargava and Prof. Gordon H. Bradley for their guidance and patience during the work in performing this research.

The author also would like to acknowledge the support of his friends Clay Tettelbach and Mehmet Bediz.

ACKNOWLEDGMENT

The author wants to thank Prof. Hemant K. Bhargava and Prof. Gordon H. Bradley for their guidance and patience during the work in performing this research.

The author also would like to acknowledge the support of his friends Clay Tettelbach and Mehmet Bediz.

I. INTRODUCTION

A. OBJECTIVES

The main objective of this research is to investigate the design and implementation of computer-based modeling systems and environments focusing on computational dependencies between model components. Another important aspect of this research is the development of a user-friendly model formulation interface for algebraic modeling languages and systems; these facilitate the description and implementation of mathematical models by allowing the modeler to employ commonly known and powerful algebraic notation instead of language specific codes. In spite of the increasing popularity of GUI-based operating systems, most of the algebraic modeling languages, today, are still file-oriented, text-based, and therefore require structured declarations and formal model definitions. The utilization of the standard graphical screen objects of a graphics-based operating system provides better visualization of models and stronger human-computer interaction.

B. BACKGROUND

1. Model Abstraction

A *Model* is a preliminary work or construction or schematic description of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics [Ref. 1]. It is often a simplified representation or description of a system or complex entity. In Figure 1, Taha [Ref. 2] presents the levels of abstraction of a real-life situation that lead to the construction of a model. Although a real situation may involve a substantial number of variables and constraints, usually only a fraction of these variables and constraints truly dominates the behavior of the real system. Thus the simplification of the real system for the purpose of constructing a model should concentrate primarily on identifying the dominant variables and constraints as well as other data pertinent to decisions making.

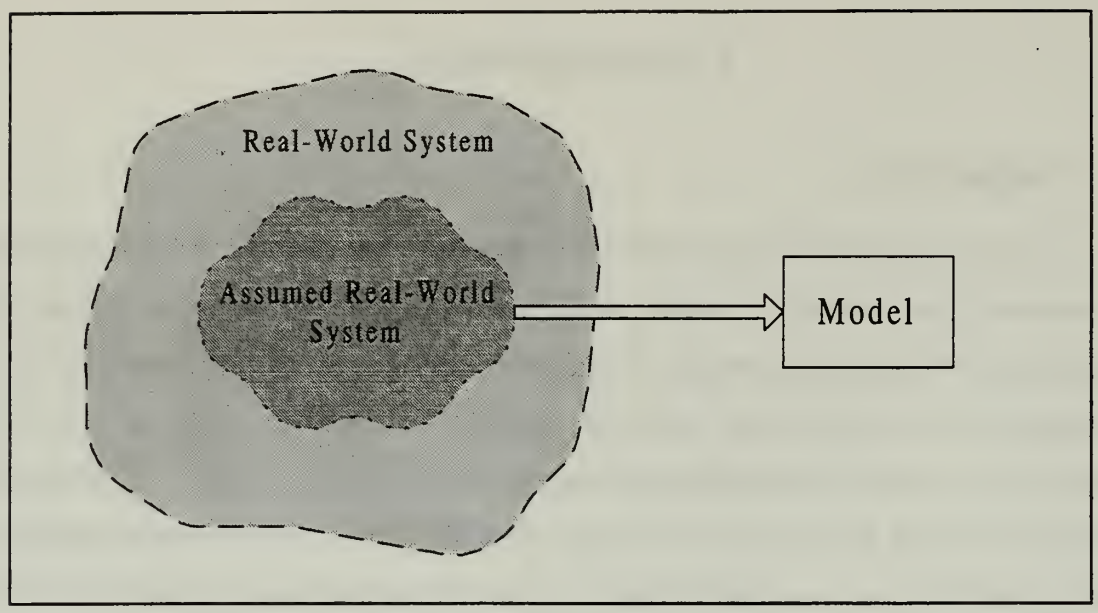


Figure 1. Levels of Abstraction

The *assumed real-world system* is abstracted from the real situation by concentrating on identifying the dominant factors (variables, constraints, and parameters) that control the behavior of the real system. The model, being an abstraction of the assumed real-world system, then identifies the pertinent relationships of the system in the form of an objective and a set of constraints.

Modeling is an indispensable activity in most organizations, whether mathematical in nature as in management science or operations research applications, or computer-oriented, as in the development of information systems [Ref. 3]. Model management is an interdisciplinary pursuit which combines elements of operations research (OR), artificial intelligence (AI), database management, management science (MS), cognitive science, and decision support (among others), each of which is an accepted discipline in its own right [Ref. 4].

2. Decision Support Systems

Decision Support Systems (DSS) are *interactive* computer based systems, which *help* decision makers utilize *data* and *models* to solve *unstructured* problems [Ref 5]. A critical component of a decision support system is the *model management system*, Figure 2, [Ref. 6]. Model management systems aim to facilitate the use of formal mathematical models for decision making [Ref. 7].

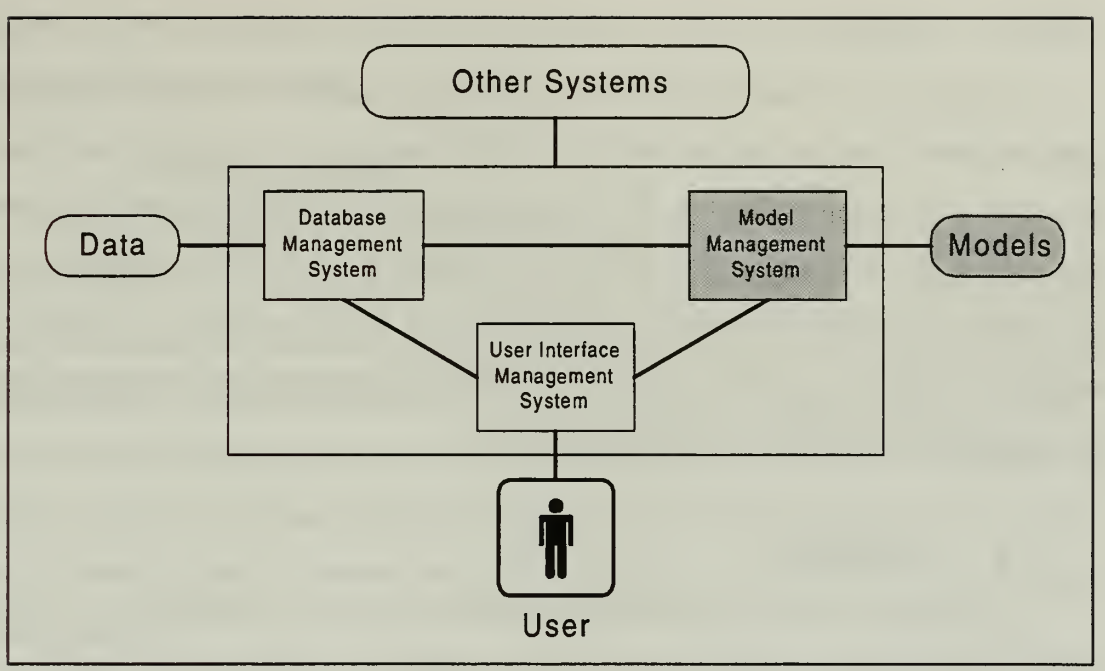


Figure 2. DSS Components

A model management system is effective when the user is able to build a model for several decision making situations, understand the model representation, retrieve information regarding the model and the decision making situation, and use the model for problem solving [Ref. 8]. Gagliardi and Spera [Ref. 9] claim that a first step toward the construction of a *good* Model Management System (MMS) is the fulfillment of some desirable features such as:

- generality, in order to encompass most of the modeling paradigm;
- independence between model representation and the appropriate solver necessary to obtain its solution;
- independence between model representation and its data;
- management of the model's life-cycle;
- high-level (intelligent) user interface.

3. Data

Models of any kind, regardless of sophistication and accuracy, may prove of little practical value if they are not supported by reliable data. In some situations, data may not be known with certainty. Rather, they are estimated by probability distributions. In such situations, it may be necessary to change the structure of the model to accommodate the probabilistic nature of the demand. This gives rise to the so-called *probabilistic* or *stochastic models* as opposed to *deterministic models*. The gathering of data may actually be the most difficult part of completing a model.

4. Graphics

According to Saaty, mathematical modeling consist of identification of variables and the dependencies among them [Ref. 10]. Dependency diagrams are useful tools to represent value dependencies between variables of the model [Ref. 11]. The advantages of using a graphical model management system over a textual system include greater flexibility and ease of use [Ref. 12].

5. Structured Modeling

Structured Modeling [Ref. 13 and 14] was developed as a comprehensive response to the perceived shortcomings of modeling systems available in the 1980s. It is a systematic way of thinking about models and their implementations that is based on the idea that every model can be viewed as a collection of distinct elements, each of which has a definition that is either primitive or based on the definition of other elements in the model. Elements are categorized into five types (so-called *primitive entity, compound entity, attribute, function,* and *test*), grouped by similarity into any number of classes called *genera*, and organized hierarchically as a rooted tree of *modules* so as to reflect the model's high-level structure. It is natural to diagram the definitional dependencies among elements as arcs in a directed acyclic graph. Moreover, this dependency graph can be computationally active because every function and test element has an associated mathematical expression for computing its value. The spirit of the research presented here

is closest to that of Geoffrion's work on structured modeling, particularly to the *genus graphs* used in structured models.

6. Modeling Languages

Modeling languages are tools for implementing large-scale iterative schemes and provide a means for connecting user's functions, solvers and interfaces. If people could deal with mathematical programs in the same way that algorithms do, the formulation and generation phases of modeling might be relatively straightforward. In reality, however, there are many differences between the form in which human modelers understand a problem and the form in which algorithms solve it. Conversion from the "modeler's form" to the "algorithm's form" is consequently a time-consuming, costly, and often error-prone procedure [Ref. 15].

Modeling languages are designed to express the modeler's form in a way that can serve as direct input to a computer system. Then the translation to the algorithm's form can be performed entirely by computer, without the intermediate stage of human translation. Modeling languages can help to make mathematical programming more economical and reliable; they are particularly advantageous for development of new models and for documentation of models that are subject to change. An *algebraic* modeling language is a popular variety based on the use of traditional mathematical notation to describe objective and constraint functions. An algebraic language provides computer-readable equivalents of notations such as $x_j + y_j$,, $\sum_{j=1}^{n} a_{ij*} x_j$, $x_j \geq 0$, and $j \in S$ that would be familiar to anyone who has studied algebra or calculus. Familiarity is one of the major advantages of algebraic modeling languages; another is their applicability to a particularly wide variety of linear, nonlinear and integer programming models. The availability of an algebraic modeling language makes it possible to emphasize the kinds of general models that can be used to describe large-scale optimization problems. While successful algorithms for mathematical programming first came into use in the 1950's, the development and distribution of algebraic modeling languages only began in the 1970's. Since then, advances in computing and computer science have enabled such languages to become steadily more efficient and general [Ref. 16]. Mathematical programming provides

a way of describing a problem and a variety of methods for solving it [Ref. 17]. However, today's modeling languages still lack many usability features of a GUI environment and require very strict syntax rules, which make them hard to learn and use.

7. Summary

This research focuses on a fairly simple, though perhaps not widely used, element in the practice of modeling: computational dependencies between model components (chiefly between model variables). It proposes that dependencies between variables be a foundation for building and using models and modeling languages. It aims to show that explicit dependency specification can support model documentation, validation, formulation, implementation, comprehension, maintenance, and reuse. That is, it would impact nearly every step of the modeling life cycle. Finally, the research describes a GUI-based prototype modeling system that is based on these ideas and can be used in conjunction with most modeling languages in use today.

C. RESEARCH QUESTIONS

With the maturity in modeling languages and solvers for mathematical models, it becomes possible to examine certain other questions in the practice of modeling:

- How can the computational dependencies between model components (mainly model variables) be used in modeling?
- Can the dependencies between variables be a foundation for building and using models and modeling languages?
- How should these models, and their assumptions, be documented and reasoned with?
- How can model structure be comprehended, preferably at a glance, without having to examine the detailed mathematical formulation (graph-based modeling)?

- Is it possible to improve usability features and ease-of-use by adding an intermediate interface between the user and the modeling languages of today?

- What other methods can be incorporated into, or used in conjunction with, existing modeling languages to make them even more powerful and useful?

D. METHODOLOGY

The initial research approach for this thesis was a combination of literature review of current books and periodicals, and site surveys of existing WWW sites and USENET newsgroups to obtain information for both requirements guidance and the DELPHI programming language.

Once the preliminary requirements were analyzed, a prototype system was developed by using *Delphi*™ programming language, which is a visual, object-oriented, component-based Rapid Application Development (RAD) and database development tool for Microsoft Windows®. In order to get feedback from end users and get them involved with the actual system design, several usability tests were performed during system development. The feedback of the end users was analyzed to refine the GUI design of the prototype.

E. CHAPTER OUTLINE

Chapter II presents how the author envisions the modeling process. After introducing some basic definitions and notations, it discusses how to formalize this process and then make it a part of the modeling system. It also bolsters these ideas with several examples in order to clarify certain concepts.

Chapter III describes the laws of indexing and the rules for using indexical structures. At the outset of the mathematical modeling development effort, these specific laws and rules would facilitate the refinement/validation of the new model (especially, during the generation of equations by using the functional forms). This chapter presents various kinds of checks, which could be performed by the system to ensure that the set of

dependencies is consistent with the set of equations, in order to make more precise equations.

Chapter IV discusses the system implementation phase of the application development. The chapter addresses issues associated with designing an appropriate user interface and selecting appropriate application development tools.

Chapter V provides a summary of the thesis and contributions of this research. It also describes some of the further research opportunities in the topic area.

II. CONCEPTUAL FRAMEWORK

This chapter presents how the author envisions the modeling process. After introducing some basic definitions and notations, it discusses how to formalize this process and then make it a part of the modeling system. It also bolsters these ideas with several examples in order to clarify certain concepts.

A. OVERVIEW

With the maturity in modeling languages and solvers for mathematical models, it becomes possible--and worthwhile--to examine the design and implementation of computer-based modeling systems and environments in the decision sciences and information systems. This paper focuses on computational dependencies between model variables. It proposes that model formulation can be supported by defining the variables and their dependencies between them. Once these model components are defined, then their utilization is pretty straightforward in conjunction with the widely used modeling languages of today.

1. Model Variables and Dependencies

Mathematical modeling/formulation often begins with the identification of relevant variables and relationships among them [Ref. 10]. Dependency graphs are often used to display dependencies. In terms of the modeling/decision life cycle, the step of problem structuring occurs before model formulation; dependency graphs are a useful tool for problem structuring [Ref. 11]. In the model management and modeling languages literature though, there are only a few papers that attempt to make use of these ideas. Dependencies are an important component of Geoffrion's structured modeling framework; genus graphs are explicitly used in the development of structured models [Ref. 14].

a. Dependency Diagrams

A computational dependency from variable **X** to variable **Y** represents the problem assumption (or model requirement) that the value of **Y** changes as a function of the change in the value of **X**. *Nodes* and *directed arcs* are the major elements of a dependency diagram. Nodes represent sets, variables, or indexed variables. Directed arcs represent dependencies between variables and the direction of dependency. Value of one variable (**Y**) changes as value of the other (**X**) changes. Figure 3 introduces the basic elements of a dependency diagram (Read as *"Y depends on X"*).

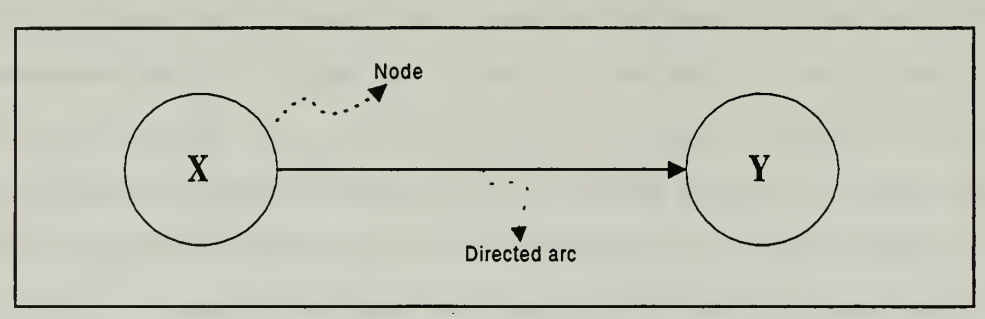


Figure 3. Elements of a Dependency Diagram

For example, since revenue depends on sales price and sales quantity, we get the dependency diagram that is shown in Figure 4.

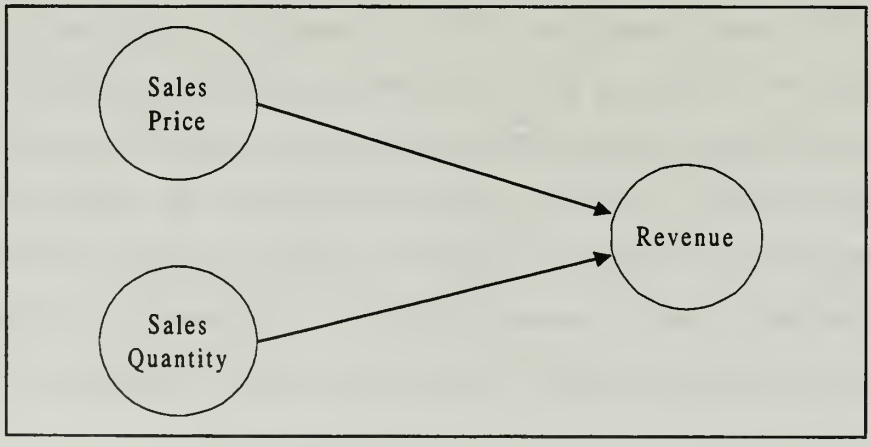


Figure 4. Dependencies

It should be evident that the set of computational dependencies for a problem is model-specific; different model formulations for a problem may have different dependencies. Two separate formulations may also have the same dependencies.

An important semantic consideration in developing dependency diagrams is to separate the dependencies that arise due to relationships between components in the model from those that arise due to the modeling task (e.g., optimization) applied to the model.

b. Dependency Types

Directed arcs can be extended to indicate the behavior (e.g., increasing/decreasing, direct/inverse, linear/nonlinear, logarithmic/exponential, etc.) and type (e.g., structural, definitional, or computational) of the dependency.

Structural and *Definitional* dependencies exist in a model definition, regardless of what task (e.g., optimization of some function, or simulation, etc.) is to be performed on that model. These are dependencies that are assumed in the problem definition. *Computational* dependencies are those that arise due to the execution of the computational task associated with the model.

Structural dependency between X and Y: When the data structure of Y is dependent on the value of X. E.g., X is a set of cities, and Y is , say, d(j, k) – the distance between cities j and k.

Definitional dependency between X and Y: When the value of Y is, by definition of X and Y, a function of the value of X. E.g., if Y is Total Cost, and X is Acquisition Cost. Or, the value of an outcome variable is a function of the values of decision variables and other input variables and uncontrollable variables.

Computational dependency between X and Y: When the value of Y is computed as the result of executing a model, with a given task, where X is one of the inputs of the model and Y is an output. E.g., the optimal value of some decision variable is a function of the various input values.

c. User Interface

Extensive research has been done in the area of model management systems [Ref. 18]. While most of the research has been directed towards model representation [Ref. 19], little effort has been directed towards the user interface of a model management system [Ref. 20]. Portability concerns also have implications for the user interface [Ref. 17]. The user interface is critical to the success of a model management system. Although highly sophisticated OR and statistical models exist today, the effective use of these models depends significantly on a good user interface [Ref. 14].

d. Model Manager

The objective of the prototype program, *Model Manager,* is to facilitate the creation, comprehension, and maintenance of mathematical models via the explicit representation of computational dependencies between variables in the model. The user of this program declares sets, variables, and test elements and then states the dependencies between them using nodes and directed arcs in a GUI screen. Then the user declares the type and the behavior of the dependency. Using these inputs, the system generates the model expressions in functional forms. The user works with these equations and completes them, if necessary. Any necessary changes are made first at the dependencies level. Various kinds of checks are performed by the system to ensure that the set of dependencies is consistent with the set of equations. By using the modeling rules (indexical equivalence, dimensional equivalence, and model structure --e.g., linear programming--), the program refines/validates functional forms in order to make more precise equations. Then the user completes/changes suggested equations. The program then checks for syntax/logic errors in order to verify the validity of the equations again. Errors in areas such as parentheses and punctuation can virtually be eliminated. By encoding the general template required by the modeling system, much of the code surrounding the actual equations should be automatically generated. Finally the program generates actual code and saves the model in the file format of the modeling language of choice or HTML document.

2. Dependencies in Modeling and the Modeling Life Cycle

Dependency diagrams that capture computational dependencies are used as a basic problem structuring tool. The idea is that an unstructured problem description can first be given some formal structure by conceptualizing and explicitly representing the dependencies between the variables of interest. In general, these dependencies are model specific and therefore influence the form of any specific mathematical model that is constructed. That is, any detailed mathematical formulation ought to be consistent with the dependencies laid out in the dependency graph. In the practice of modeling, however, dependency graphs are rarely used and no modeling language provides features for explicit representation and use of dependency graphs.

We believe that dependency graphs can play a useful role in model management. They serve not only the model formulation phase by representing the qualitative problem in a more structured way, but also serve as model documentation once the model is specified mathematically. They also enhance model comprehension, which should facilitate model maintenance and model reuse. Automatic validation can be performed. The documentation is active; that ensures it will be used. Further, it does not just act as a guard; it actually can help in the development of the model specification. That is, the "code" that represents the model code can be partially generated based on the dependency graph.

a. Dependency Diagram: EOQ Example

Consider the well-known lot size inventory management model. The input variables are D, h, c, and A; output variables are q, n, and TC. Mathematically, the relevant equations in the model are:

$$TC = \frac{A.D}{Q} + \frac{h.Q}{2} + c.D$$

$$Opt(Q) = Q^* = \sqrt{\frac{2.A.D}{h}}$$

$$n = \frac{D}{Q} \qquad \text{and} \qquad Opt(n) = \frac{D}{Opt(Q)}$$

TC: Total cost	n: Number of orders
A: Setup cost per order	h: Holding cost
D: Demand	c: Unit cost
Q: Order quantity	Q^*: Optimal order quantity

b. Dependencies in the EOQ Model

The dependencies between model components are shown in Figure 5.

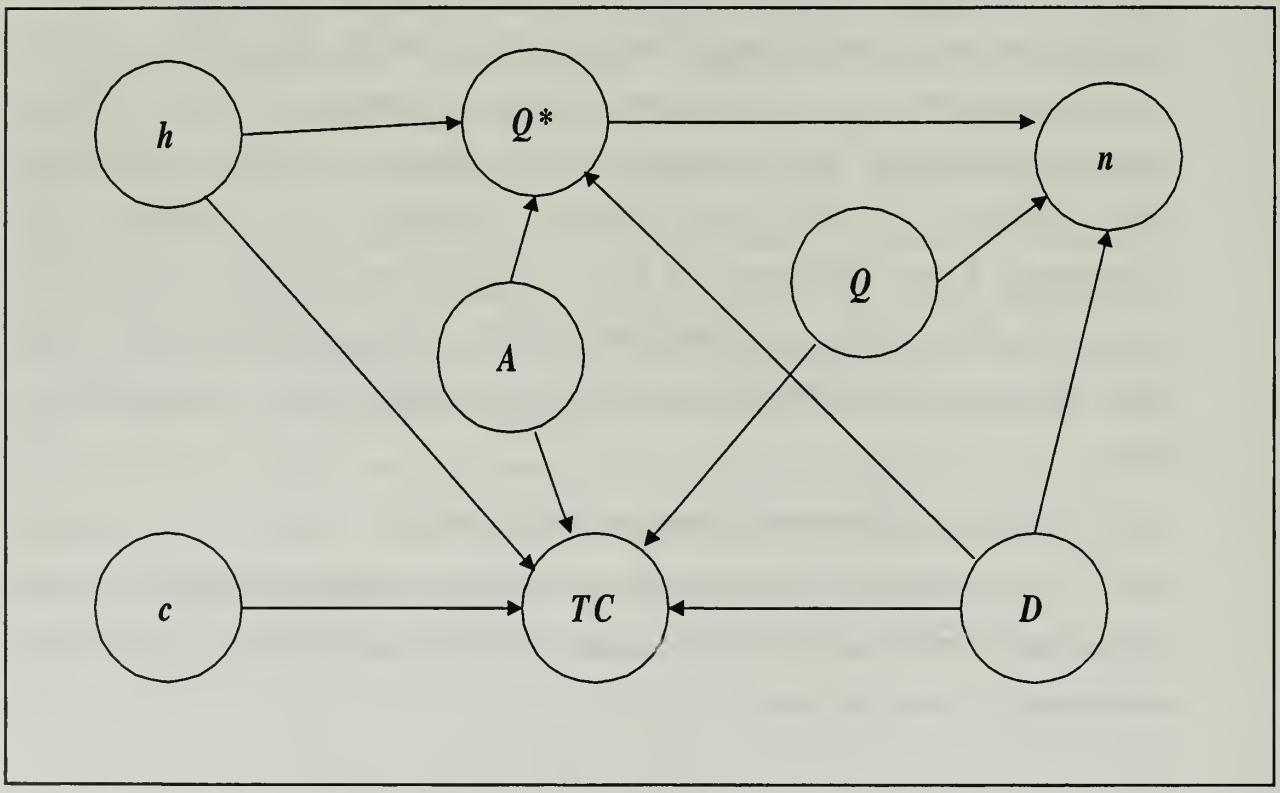


Figure 5. Dependency Diagram for the EOQ Model

Or textually;

TC depends on c	Q^* depends on h
TC depends on h	Q^* depends on A
TC depends on A	Q^* depends on D
TC depends on D	n depends on D
TC depends on Q	n depends on Q^*
	n depends on Q

It is easy to extend this representation. For example, suppose one wanted to add a new output variable S (total setup cost);

$$S = n \,.\, A$$

then, the following two dependencies must be added:

S depends on A

S depends on n

Consider, for the moment, that only the various dependencies have been defined in developing the model so far; i.e., the exact equations have not been developed. Let us examine how a modeling system centered around the idea of a dependency graph can assist in the development and management of a model for this problem.

c. Generation of Model Expressions (Functional Form)

First, we argue that from the dependencies given above, we --or a computer program-- should be able to generate the following equations:

$$S = f_1 (n, A)$$

$$n = f_2 (D, Q)$$

$$TC = f_3 (A, D, Q, h, c)$$

$$Q = f_4 (A, D, h)$$

where f_1 , f_2 , f_3 and f_4 are some mathematical functions.

Of course, the formulation is incomplete, yet it provides certain function forms that any complete model should be consistent with. If we were to combine dimensional information, we can make the equations more precise; alternately we could use that information to validate the functional form developed by the user. The dependency diagram could be used to convey the model structure and assumptions at a glance. Therefore, even for this simple problem, dependencies could be useful in model formulation, validation, comprehension, and reuse.

d. Dependency Diagrams: Transportation Model

Now consider a more interesting example, one that makes use of index sets. Now we examine another well-known model, Transportation Model. The following variables are defined in this model, the last two being test variables that represent the constraints in the model.

Cap (i) : Supply capacity at source *i*

Dem (j) : Demand at destination *j*

Out (i) : Total actual flow out of source *i*

In (j) : Total actual flow into destination *j*

Flow (i, j) : Actual flow from *i* to *j*

Cost (i, j) : Cost of unit flow from source *i* to destination *j*

TC : Total cost for all flows

Test elements that represent the constraints in the model:

T_1 *(i)* : Capacity test for source *i*

T_2 *(j)* : Demand fulfillment test for destination *j*

e. Dependencies in the Transportation Model

By examining the standard problem definition, we are able to specify the following dependencies between the variables. Textually:

TC depends on *Cost (i, j)*

TC depends on *Flow (i, j)*

In (j) depends on *Flow (i, j)*

Out (i) depends on *Flow (i, j)*

T_1 *(i)* depends on *Cap (i)*

T_1 *(i)* depends on *Out (i)*

T_2 *(j)* depends on *Dem (j)*

T_2 *(i)* depends on *In (j)*

f. Generation of Model Expressions (Functional Form)

From this we should be able to derive the following equations:

$TC = f_1 (Cost (i, j), Flow (i, j))$

$In (j) = f_2 (Flow (i, j))$

$Out (i) = f_3 (Flow (i, j))$

$T_1 (i) = g_1 (Cap (i), Out (i))$

$T_2 (i) = g_2 (Dem (j), In (j))$

where, once again, f_1, f_2, and f_3 are some mathematical functions, and g_1 and g_2 are conditional expressions with the same truth value as T_1 and T_2. Again, these dependencies could be useful in model formulation, validation, comprehension, and reuse.

g. Generation of Equational Forms Using Indexical Rules

However, more can be achieved by exploiting the indexical information contained in the model. This information is combined with a model formulation rule that requires model expressions to be indexically homogeneous. That is, all terms in an expression must have the same indexical (where the indexical is defined as the unbound index variables in the expression). Assuming for now that the only indexical operation is the summation (Σ) operator, it is easy to see that the equations above can be further specified as shown below:

$$TC = \sum_i \sum_j f(Cost(i,j), Flow(i,j)) \qquad \text{(Eq. 2.1)}$$

or

$$TC = f\left(\sum_i \sum_j Cost(i,j), \sum_i \sum_j Flow(i,j)\right) \qquad \text{(Eq. 2.2)}$$

$$In(j) = \sum_i Flow(i,j) \qquad \text{(Eq. 2.3)}$$

$$Out(i) = \sum_j Flow(i,j) \qquad \text{(Eq. 2.4)}$$

$$T_1(i) = Cap(i) \ \ \textbf{OP} \ \ Out(i) \qquad \text{(Eq. 2.5)}$$

$$T_2(i) = Dem(j) \ \ \textbf{OP} \ \ In(j) \qquad \text{(Eq. 2.6)}$$

where ***OP*** is some comparison operator.

It may be noticed that Eq. 2.3 and Eq. 2.4 are completely specified in this manner. By considering additional information --such as the type model being developed (a linear program)-- the equational forms can be further constrained and made more complete. The goal, though, is not to automate the development of the mathematical formulation on the basis of the dependency diagram. In general, there may still be far too many possible formulations. The point is that these formulations must all satisfy a variety of constraints that derive from the dependency diagram.

3. Dependencies in Other Domains

Diagrams are used to facilitate problem solving in mathematics, physics, chemistry, engineering, economics, business, and other scientific areas. Research indicates that users understand and use diagrams easily and more efficiently than textual comprehension. Graphical representations help users structure the problem better [Ref. 21]. Diagrammatic representations meet the criteria of descriptive and procedural adequacy for many types of problems and diagrams have traditionally been used in certain types of problem solving [Ref. 22]. Larkin and Simon [Ref. 23] show that diagrams can be searched more efficiently and need less storage for representation. Advances in the scientific world without the use of diagrams would be difficult [Ref. 22].

Computer science, for example, makes use of diagrammatic approach in many areas extensively. Entities and their dependencies are often presented with diagrammatic representations in various contexts. *Entity relationship* (ER) diagrams and *semantic object models* (SOM) in database design, *state diagrams* in automata theory, *neural networks* in artificial intelligence (AI), *object model, dynamic model, and functional model* representations in object modeling technique (OMT), *network diagrams* in computer networks, *data flow diagrams* (DFD) and *structure charts* in structured systems design, and *traceability model diagrams* in requirements engineering are some of the most common ones. Most of the *code generators* utilize graphical representations. Dependency graphs are also used as intermediate representations in optimizing *compilers* and *software engineering*. Drawings of compiler data structures such as syntax trees, control flow

graphs, dependency graphs, are used for demonstration, debugging and documentation of compilers.

B. SUMMARY

Dependencies between model variables, as represented in dependency diagrams, are a tool for problem structuring and are a foundation for development of a formal mathematical model. They are often used, informally, by model builders in the early stages of model formulation. They are sometimes taught in courses on modeling, occasionally employed in modeling textbooks, and almost nonexistent in a formal sense in modeling languages. If incorporated into modeling languages and systems, they could be useful in model formulation, validation, maintenance, and reuse.

However, experience with similar situations in other domains (such as software development) indicates that model builders will usually not record the dependencies until they are forced to, or unless recording those dependencies provides a more tangible reduction in the model specification effort. It is, of course, additional work to document these dependencies [Ref. 24].

From the functional/equational form developed above, it is easy to write general purpose programs that will generate actual code in the modeling language of choice. Further, by encoding the general template required by the modeling system, much of the code surrounding the actual equations could be automatically generated. Errors in areas such as parenthesis and punctuation can virtually be eliminated. Thus, material support can be given to the model builder who would not have to develop the entire model from scratch by hand. We believe, this would be instrumental in encouraging modelers to conceive of models in terms of dependencies, document this conceptualization formally and explicitly, and use it subsequently in model formulation and maintenance.

III. REASONING WITH DEPENDENCIES

This chapter describes the laws of indexing and the rules for using indexical structures. At the outset of the mathematical modeling development effort, these specific laws and rules facilitate the effort to refine/validate the new model (especially, during the generation of equations by using the functional forms). Based on the ideas of Bhargava [Ref. 25], this chapter presents various kinds of checks, which are performed by the system to ensure that the set of dependencies is consistent with the set of equations.

A. OVERVIEW

Index sets, indexing of variables over these sets, and expressions containing indexing structures, are the most important elements of mathematical modeling. The laws of indexing and rules for using indexical structures are usually straightforward and rarely explicitly stated. However, with the advent of modeling languages that allow indexed expressions, it is important to state the semantics of indexing operations and expressions in a clear, implementation-independent manner.

Indexical reasoning in modeling languages can be used for;

- Validating expressions involving indexed variables, indexing structures, and indexical operators,
- Constraining the formulation of expressions in a way that it satisfies indexing laws,
- Deducing expressions that satisfy all laws.

B. FRAMEWORK

1. Objects, Terminology, and Notation

The following examples illustrate the framework, concepts and objects relevant to indexical reasoning and how these are related.

a. Example 1 (Assignment Constraint)

$$assign(i \in I): \sum_{j \in J} x_{i,j} = 1$$

This constraint states that each individual (in the set I) must be assigned exactly one job (from set J). There are two *index sets* (I and J), and one *indexed variable* ($x_{i,j}$) with *index symbols* (or indices) i and j being used to range over sets I and J respectively. The *test element* assign (i ∈ I), indexed over set I, returns a value "true" when the mathematical condition is satisfied. The symbol Σ represents the *indexical operator* for summation, and $j \in$ J is an *indexing structure*.

An important reason to distinguish between the index set itself and the index symbols used in writing expressions that range over the set, is that an indexed variable may be indexed over a set more than once. For example, consider the subtour elimination constraint in a traveling salesman problem.

b. Example 2 (Subtour Elimination Constraint)

$$subtour(j \in J, k \in J): u_j - u_k + n \cdot y_{j,k} \leq n - 1$$

Here, the indexed variable $y_{j,k}$ represents travel between nodes j and k, where both j and k range over the same set of nodes J. The variable n represents the number of nodes; u_j and u_k are dummy variables, with the index symbols j and k again ranging over the same set J.

In general, the objects relevant to indexical reasoning in mathematical modeling are: index sets, index symbols, variables (including indexed variables) and test elements (including indexed test elements), and indexical operators. We will also define a function --the *indexical*--over the indexed expressions in the model. For now, we use our example to explain what this means: the indexical of $x_{i,j}$ is $\{i, j\}$; the indexical of $\sum_j x_{i,j}$ is $\{i\}$; the indexical of assign($i \in$ I) is $\{i\}$; and the indexical of n is $\varnothing$.

Since our discussion of these objects is largely at the meta-level, we need certain additional notation to the *sorts* (or sets) of these objects and members of these meta-level sets. In Table I, let Γ denote the set of index sets and γ denote any member of Γ; Ω denotes the set of index symbols, and ω its members; Φ denotes the set of variables and ϕ its members; Ψ denotes the set of test elements and ψ a member of Ψ. Let Λ denote the set of indexed expressions and λ a particular expression. Let Θ denote the set of indexical operators, and θ its members. Finally, let Δ denote the power set of index symbols, and δ its elements (i.e., subsets of Ω).

OBJECT TYPE	**NOTATION FOR**	
	SORT	***MEMBER***
Set	Γ	γ
Index Symbol	Ω	ω
Variable	Φ	ϕ
Test Element	Ψ	ψ
Indexed Expression	Λ	λ
Indexed Operator	Θ	θ
Power set of Ω	Δ	δ

Table I. Objects and Notations

2. Indexical Reasoning

We now define the indexical function $\iota : \Lambda \mapsto \Delta$. That is, for each indexical expression λ, its indexical –the set of index symbols that are *free* in that expression--is a subset δ of index symbols. The semantics of this function are defined in two parts: first, for the primitive expressions (variable and test elements) and second for the compound expressions involving these objects.

The indexical of a variable is simply the set of index symbols used in defining it. The indexical of a test element is, similarly, the set of index symbols used in defining it. The indexical of compound expressions is defined in the following rules:

$$\iota(\lambda_1 + \lambda_2) = \iota(\lambda_1) \quad \text{iff} \quad \iota(\lambda_1) = \iota(\lambda_2) \qquad \text{(Eq. 3.1)}$$

$$\iota(\lambda_1 - \lambda_2) = \iota(\lambda_1) \quad \text{iff} \quad \iota(\lambda_1) = \iota(\lambda_2) \qquad \text{(Eq. 3.2)}$$

$$\iota(\lambda_1 * \lambda_2) = \iota(\lambda_1) \cup \iota(\lambda_2) \qquad \text{(Eq. 3.3)}$$

$$\iota(\lambda_1 / \lambda_2) = \iota(\lambda_1) \cup \iota(\lambda_2) \qquad \text{(Eq. 3.4)}$$

$$\iota(\lambda_1^{\lambda 2}) = \iota(\lambda_1) \cup \iota(\lambda_2) \qquad \text{(Eq. 3.5)}$$

$$\iota\left(\sum_{\delta} \lambda_1\right) = \iota(\lambda_1) \setminus \delta \qquad \text{(Eq. 3.6)}$$

$$\iota(\psi : \lambda) = \iota(\psi) \text{ iff } \iota(\psi) = \iota(\lambda) \qquad \text{(Eq. 3.7)}$$

3. Dimensional Balance

Dimensional arithmetic, or the calculus of dimensions, involves operations on dimensions analogous to the arithmetic operations on numbers. Transformations of units of measurement are required in model solution and model integration. Dimensional simplification and verification of dimensional consistency of expressions is useful in model formulation and model validation [Ref. 26].

Bradley and Clemence [Ref. 27] states that one aspect of model development that has received little automated support is the verification that the algebraic representation of the model correctly represents the modeler's intention. This intention is expressed in the form of explanatory descriptions which are associated with each numeric-valued symbol in the model. These descriptions are a necessary part of any modeling effort because they assign real world meaning to the model data and computation results. The computer manipulates numbers -the meaning assigned to the data and the results is the responsibility of the modeler who is doing a "dimensional" check of each model function and constraint. This is done by replacing each numeric-valued symbol with its explanatory description and then applying two kinds of dimensional calculus. One is the calculus of measurements units: multiplication and division of units and a unit analysis to verify that pure numbers

that are added, subtracted or compared have the same scale of reference. The other kind is concerned with what the symbol represents (e.g., apples, cars) and which of its properties are being measured (e.g., cost, height, weight/time).

Bhargava [Ref. 26] presents the laws of dimensional consistency and dimensional arithmetic. The following sub-sections will explain these laws.

a. Laws of Dimensional Consistency

The laws for obtaining *dimensionally consistent* (d.c.) expressions are stated below:

- Two functional expressions may be added or subtracted only if they are *dimensionally equivalent* (d.e.). (Expressions of the form $\phi + \psi$, $\phi - \psi$ are d.c. iff ϕ and ψ are d.e.)
- Two functional expressions may be compared for equality or inequality (resulting in a conditional expression) only if their dimensions are equivalent. (Expressions of the form $\phi = \psi$, $\phi < \psi$, $\phi > \psi$, $\phi \leq \psi$, and $\phi \geq \psi$ are d.c. iff ϕ and ψ are d.e.)
- Two functional expressions may be multiplied irrespective of their dimensions. (Expressions of the form $\phi * \psi$ are d.c. if ϕ and ϕ are d.c.)
- Any dimensionally valid functional expression can be inverted. (Expressions of the form $1 / \phi$ are d.c. if ϕ is d.c.)
- The exponent of a functional expression must be dimensionless. (Expressions of the form ϕ^{ψ} are d.c. only if ψ is dimensionless.)
- The exponent of a functional expression can be fractional only if
    * each fundamental unit in the functional expression has a power that is a multiple of the inverse of that fraction, or
    * the functional expression is dimensionless. (Expressions of the form ϕ^{ψ} are d.c. if the fundamental units of ϕ^{ψ} have integer powers, i.e., ψ is an integer, or if ϕ is dimensionless, or if each fundamental unit in ϕ has a power that is a multiple of $1 / \psi$.)

- Functions which can be expressed as power series (e.g., trigonometric functions, hyperbolic functions) can be applied only to dimensionless expressions.

b. Laws of Dimensional Arithmetic

The laws of dimensional arithmetic are slightly different from that of standard arithmetic:

- The dimensions of the sum (or difference) of two expressions is the same as the dimension of either of them if the two expressions have equivalent dimensions. Otherwise, it is not defined (see laws for dimensional consistency).
- The dimension of the product (quotient) of two expressions is the product (quotient) of the dimensions of the two expressions. A dimensionless expression has dimension u_0 (= 1) which is an identity for dimensional multiplication.
- The dimension of the power of an expression is the power of the dimension of the expression.
- Any function of expression with dimension u_0 yields an expression of dimension u_0.

In their research, Bradley and Clemence [28] presents these principles:

- The product or ratio of valid types is also a valid type;
- For the operations of addition, subtraction, comparison ($\leq$, =, $\geq$), assignment, input, and output, both operands must have the same concept, quantity, and unit;
- When simplifying expressions and considering automatic conversions, concepts are checked first, then quantities, and finally units.

4. Model Type

The functional/equational forms can be further constrained and made more complete by considering additional information about the model type. For example, if the

type is a linear programming model, then we can conclude that only summation or difference signs will be used in equational forms. Similarly, if it is a non-linear program, then we can easily rationalize that product or quotient signs will be used in the model.

IV. IMPLEMENTATION

This chapter discusses the system implementation phase of the application development. The chapter addresses issues associated with designing an appropriate user interface and selecting appropriate application development tools.

A. APPLICATION DEVELOPMENT TOOL – BORLAND'S DELPHI™

The visual development methodology approach uses a visual object-oriented rapid application development (RAD) tool, in this case Borland's Delphi™, to quickly develop an application prototype.

B. APPLICATION IMPLEMENTATION

1. Functional Description of the Program

Model Manager is composed of six sub-pages. The program interface is in a tabular form, labeled with each specific function. The "tabbed notebook" approach helps the user easily navigate between the pages. Commonly-used words for menu items are used in a standard order (i.e., File, Edit, View, Run, Help) to improve system usability. The same idea is used during the construction of all the other windows, pop-up/pull-down menus, and dialog boxes.

2. Program Interface

The "About" page, depicted in Figure 6, is the beginning page of the program.

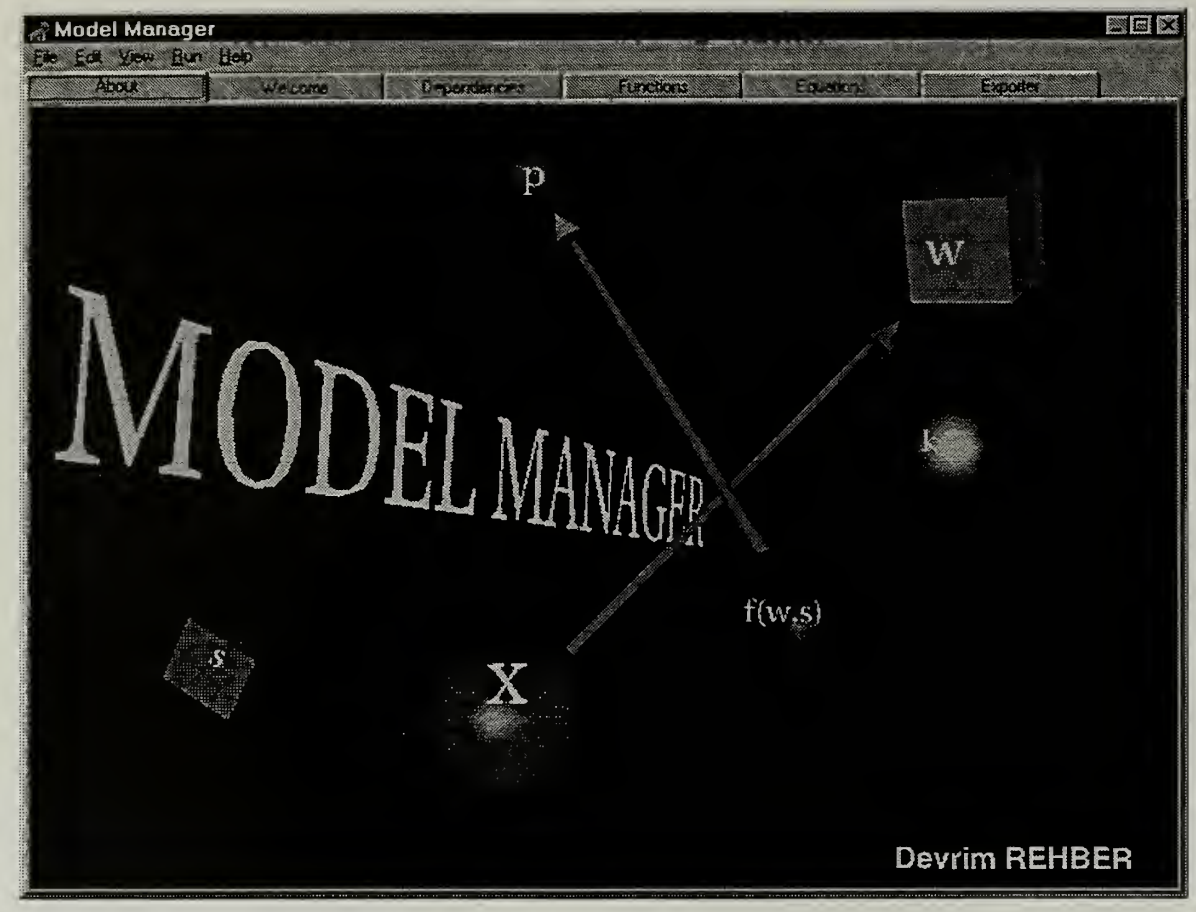


Figure 6. About Screen of Model Manager

When the user selects the "Welcome" tab of the program, the related welcome screen, Figure 7, appears. It is an introduction page which describes what the program is. The "Next >" button in this screen brings the "Model Information" message dialog box as shown in Figure 8. Here, the user defines the model and gives a name to it. The input fields in this page are optional. The modeler enters these information just for his/her own records. An alternative to using mouse or tab keys is to use shortcut keys (i.e., Alt+m and Alt+d). The same action can be done more than one way, which is an important usability issue for advanced users. This functionality is used very often throughout the program. The "Next >" button in this page submits the information inputs and opens the "Dependencies" page.

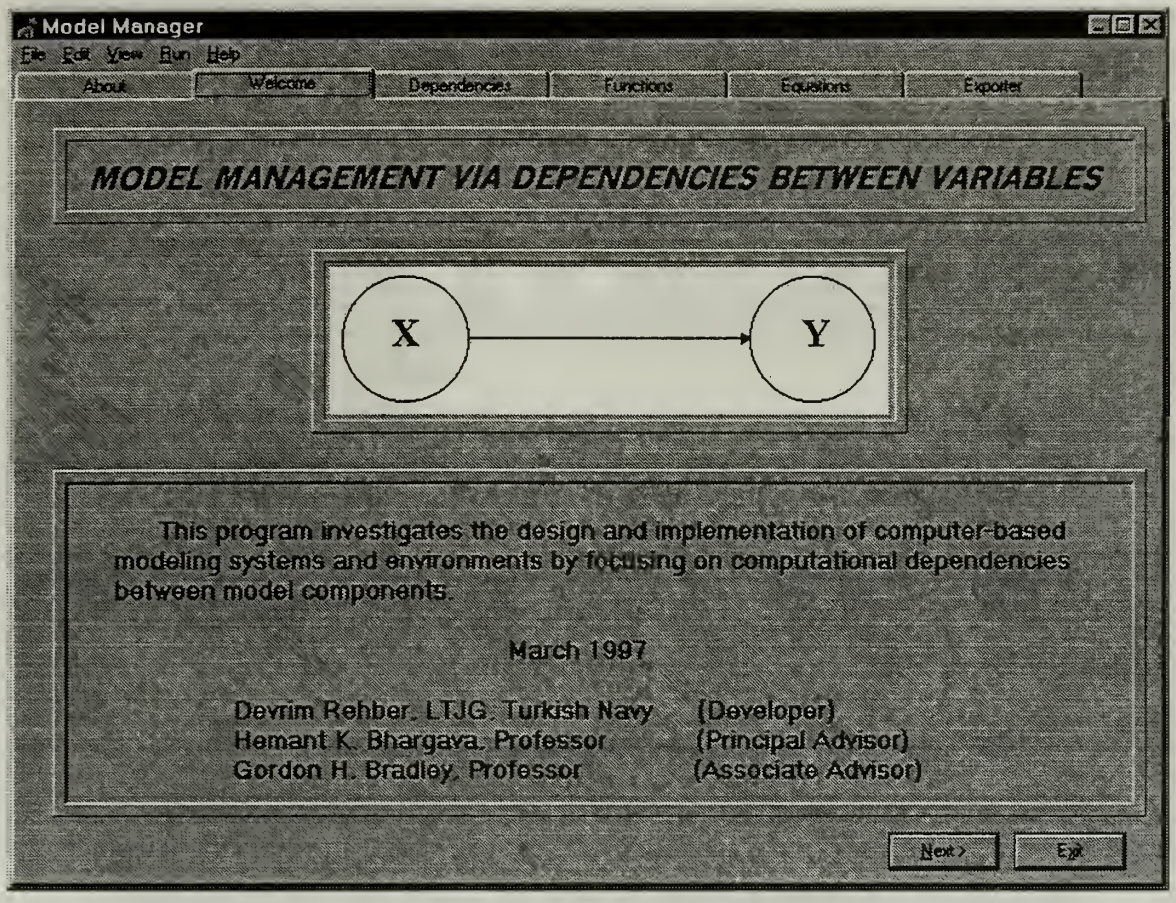


Figure 7. Welcome Page

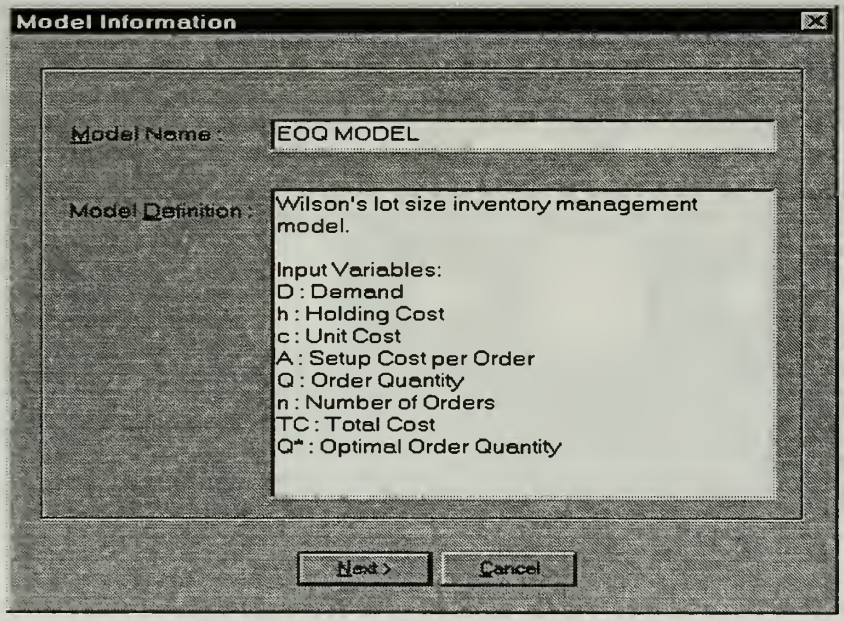


Figure 8. Model Information Screen

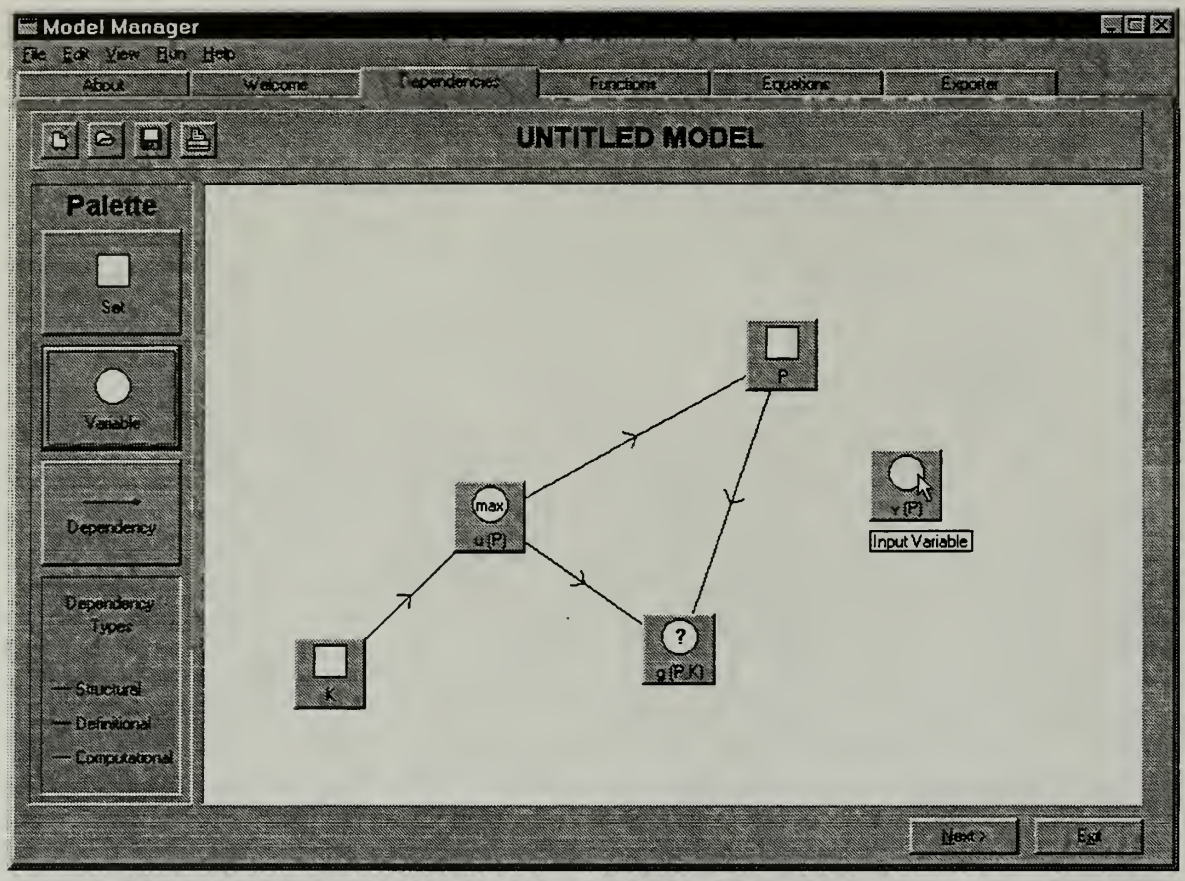
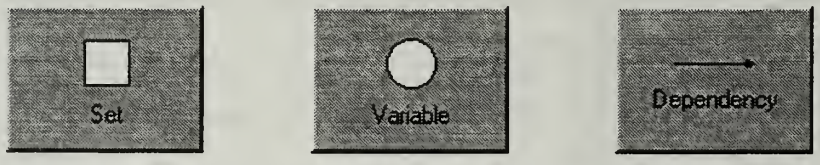


Figure 9. Dependencies Page

Figure 9 shows the dependencies page. This is where the user constructs his/her model. On the left side of the screen there is a "palette". The user chooses the model components from this palette and defines them. There are three components in the palette:



When the user selects any one of the components, an associated input box is presented. Figure 10, 11, and 12 show these input boxes respectively.

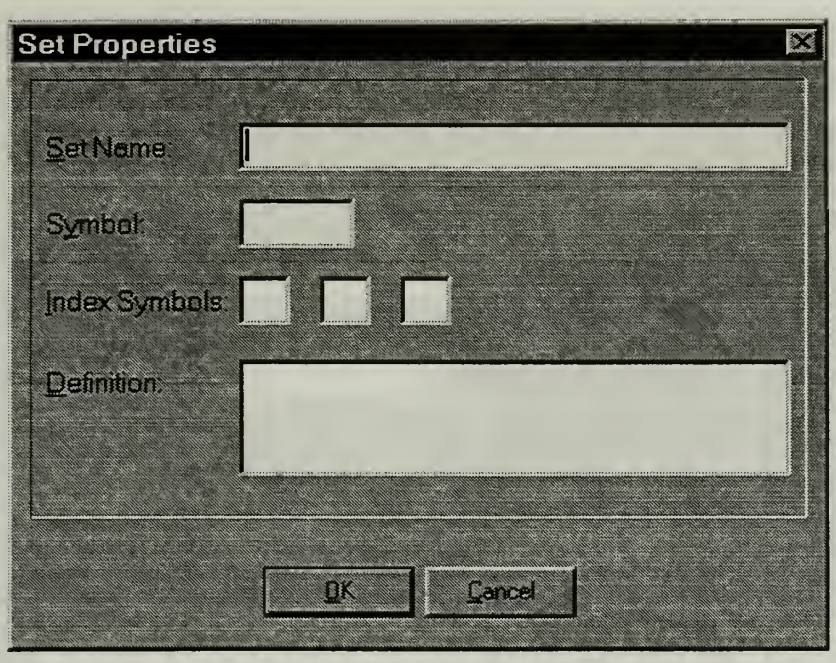


Figure 10. Input Box for Set Properties

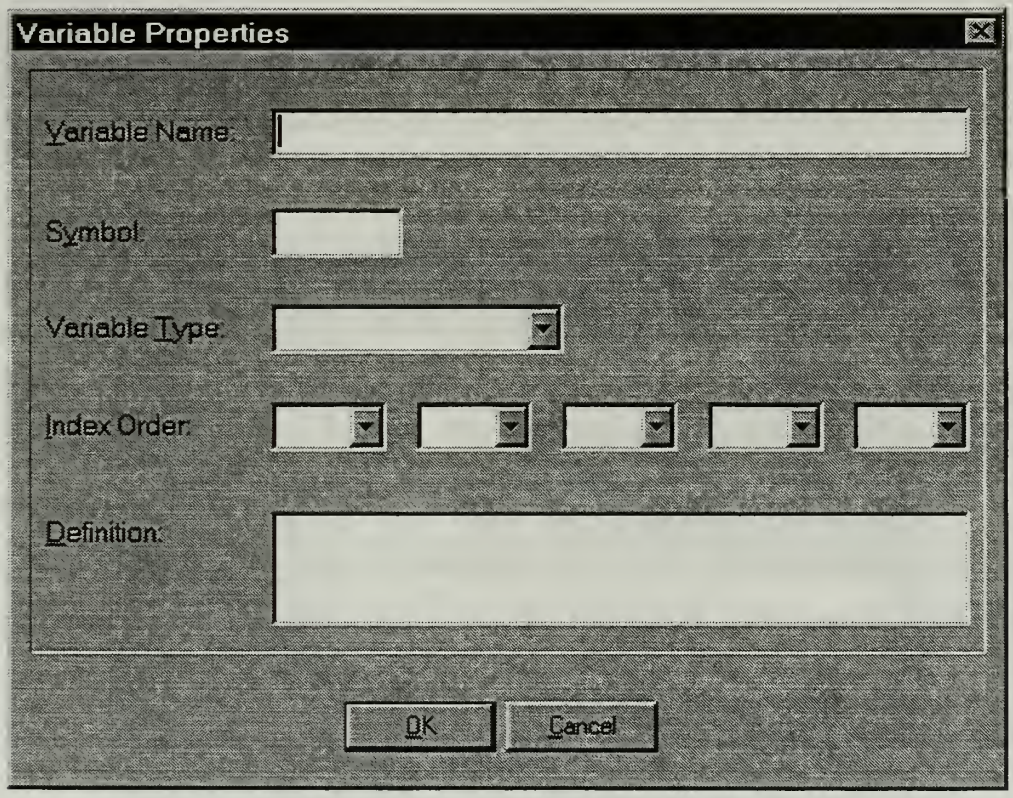


Figure 11. Input Box for Variable Properties

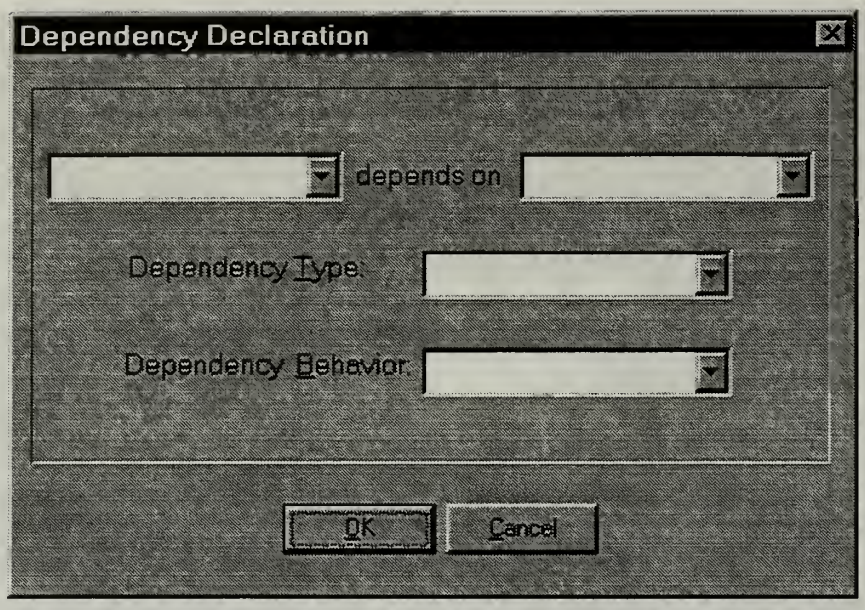


Figure 12. Input Box for Dependency Declaration

The modeler enters the values for each kind of component and then presses "OK" button in order to submit the information to the database and close the input box. On the dependencies page (Figure 9), the user chooses an appropriate space and places the component by left-clicking the mouse. The user can move the component by holding down the left mouse button and dragging to some other coordinate; or he/she can right-click the mouse button on that component to see its properties as shown in Figure 13 and 14. If the modeler wants to remove any component from the model, then he can click the right mouse button and select "Delete". Then a confirmation message dialog box appears as depicted in Figure 15; this protects the modeler from mistakenly deleting a node.

Another functionality of the right mouse button is clicking it on an empty area of the dependencies page, which brings a pop-up menu shown in Figure 16. By using this menu, the user can reach the lists of the previously created components. Figure 17, 18, and 19 show these view-only lists of databases. The other items in that pop-up menu are Clear, Print, and Help.

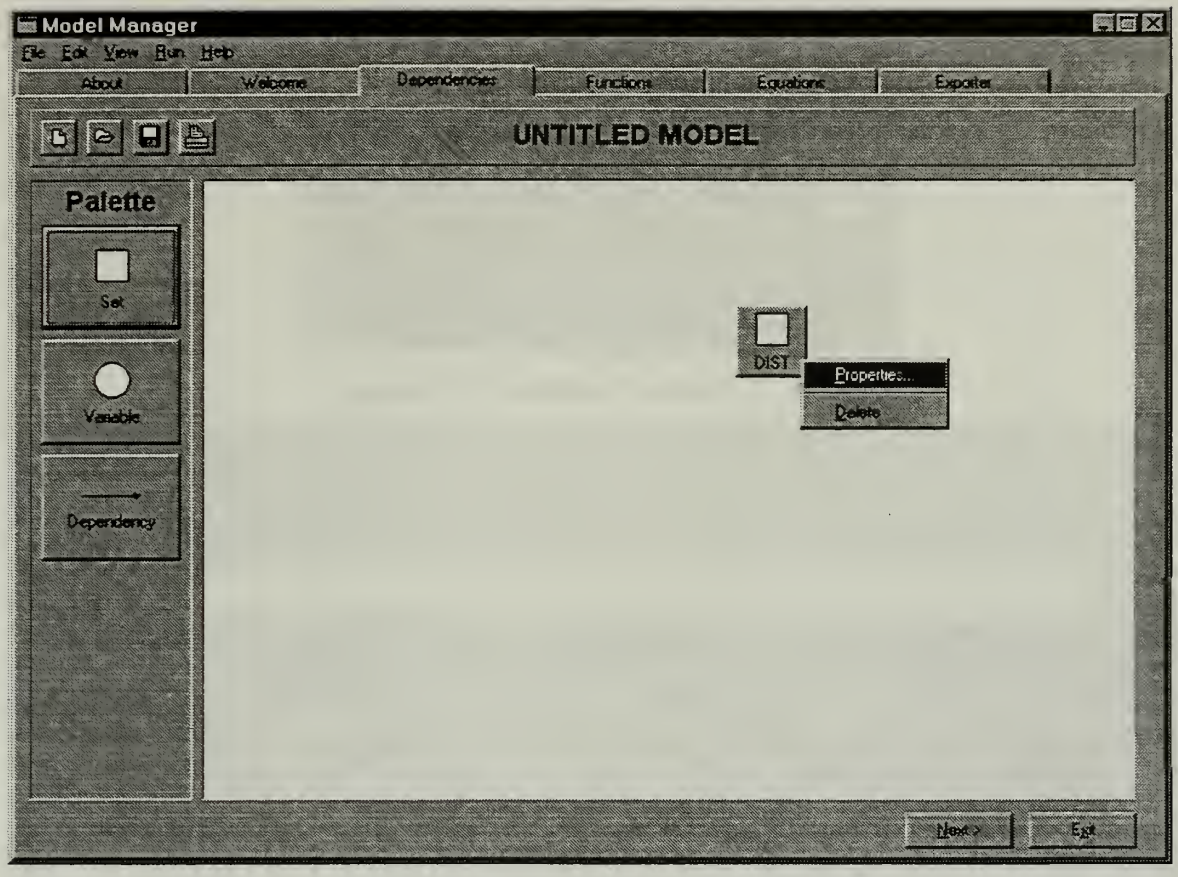


Figure 13. Right-Click on a Component

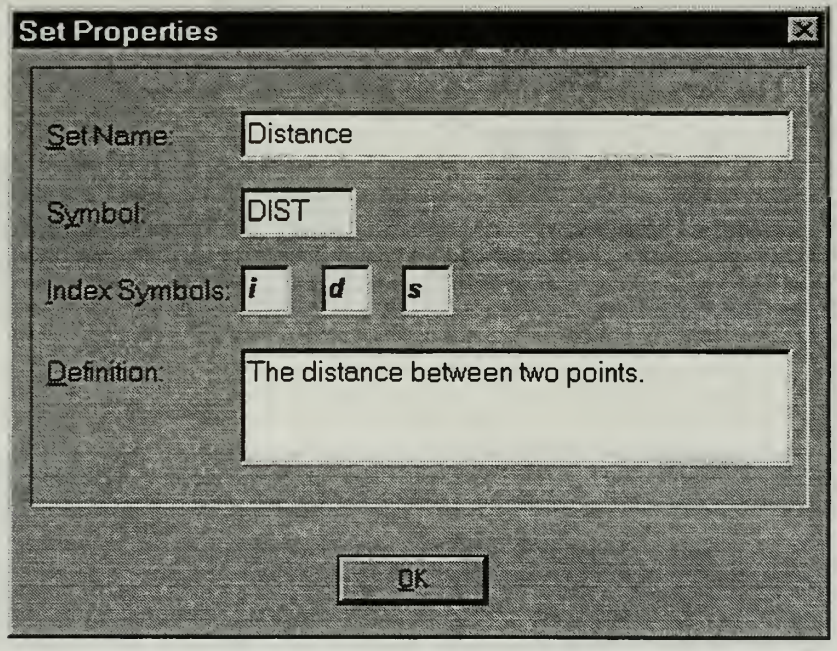


Figure 14. Properties Message Box

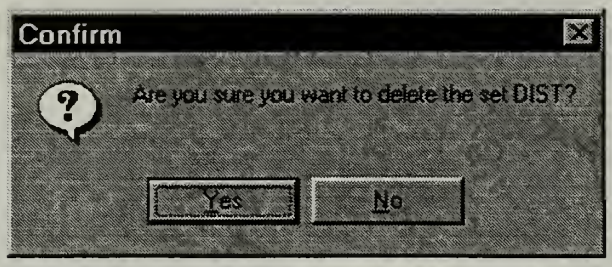


Figure 15. Delete Confirmation Message

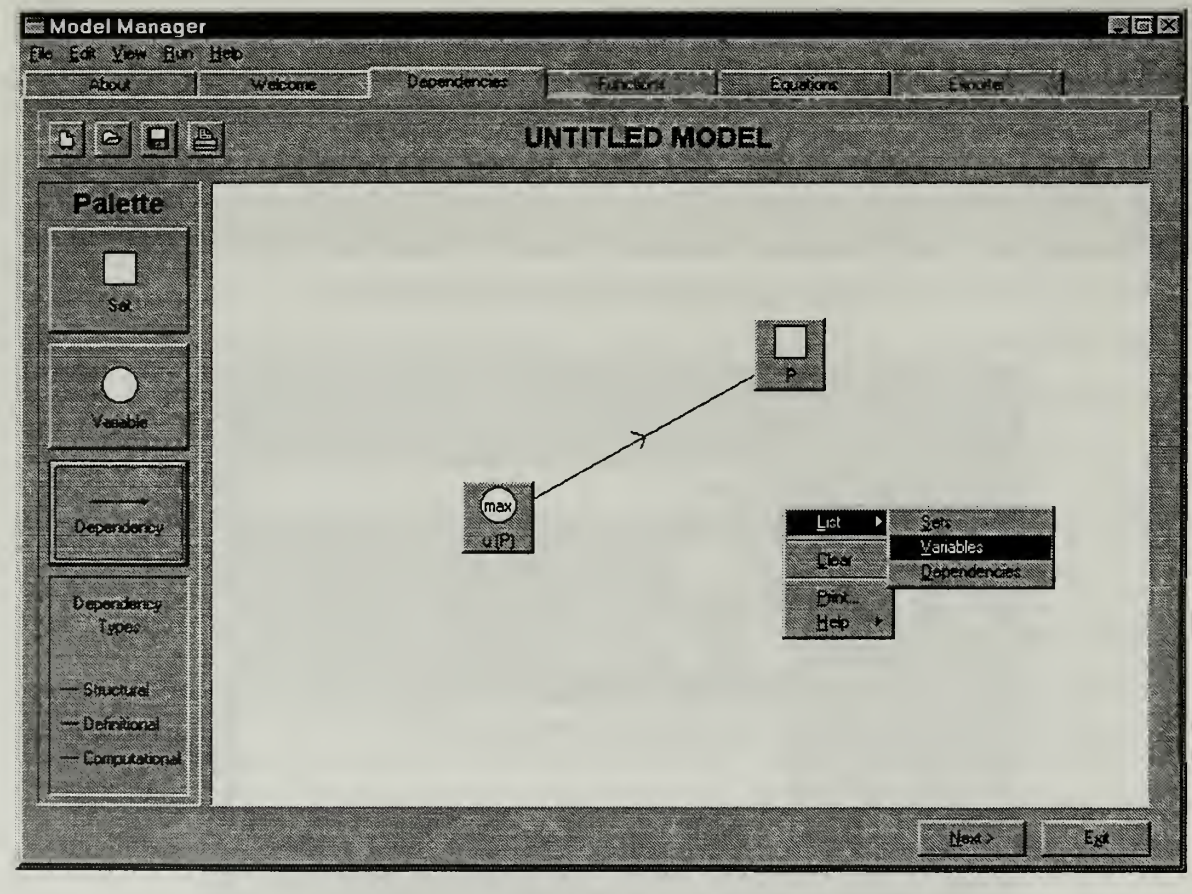


Figure 16. Right-Click on an Empty Area

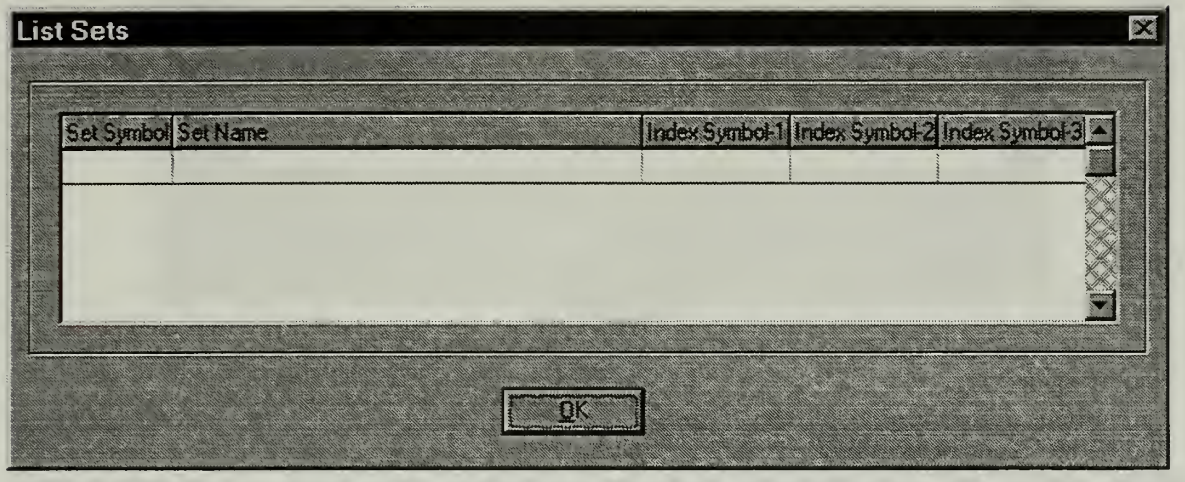


Figure 17. Message Box for Listing the Sets

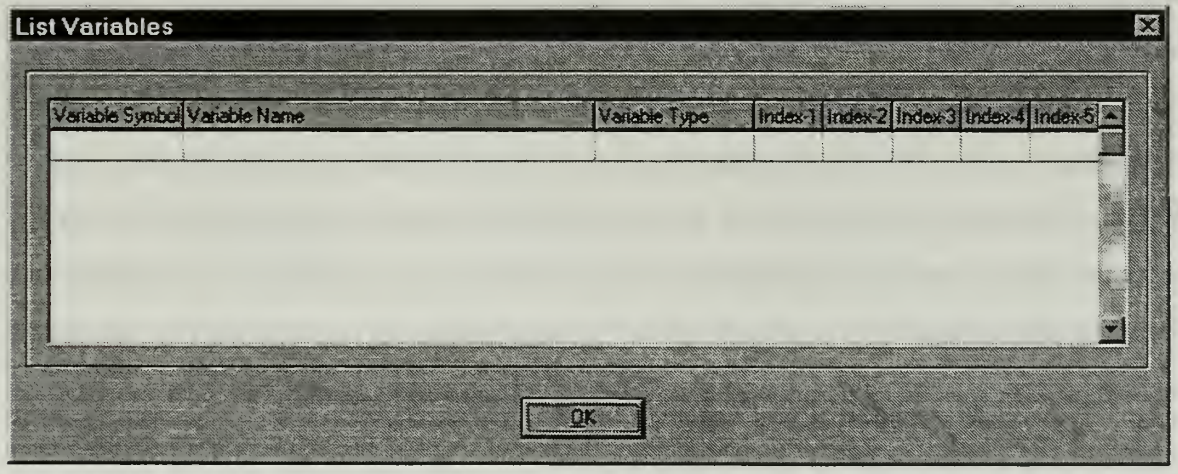


Figure 18. Message Box for Listing the Variables

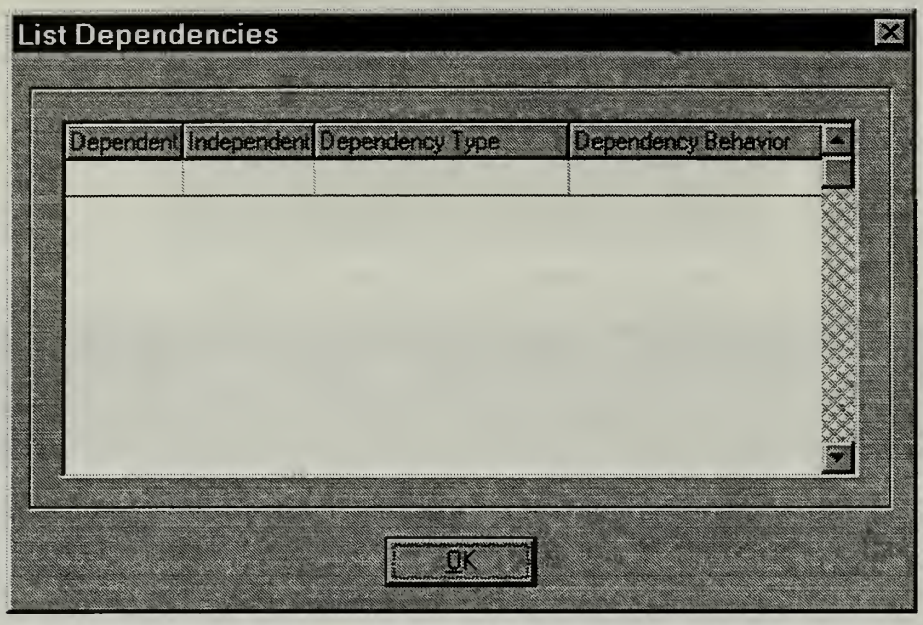


Figure 19. Message Box for Listing the Dependencies

The different types of dependencies are represented with different color codes in the model: red for structural, blue for definitional, and black for computational dependency. These color codes and hints on almost every icon and model component help the user see the dependencies easily and visualize the whole model more clearly.

After constructing the model, the user presses the "Next >" button and launches the "Functions" page. At this stage, the modeler can see his/her model in computer-generated functional form, Figure 20. He/she examines the functions, works with them, and if necessary completes them. Any changes have to be made first at the dependencies level. The functions page is a simple editor, which includes the basic editing functions. The modeler can use the icons, the shortcut keys, or right-click of mouse in order to make necessary changes in this editor. The user can also go back to dependencies page and make changes there, if necessary. Then he/she has to press the "Next >" button in that page in order to create the updated version of the functional forms.

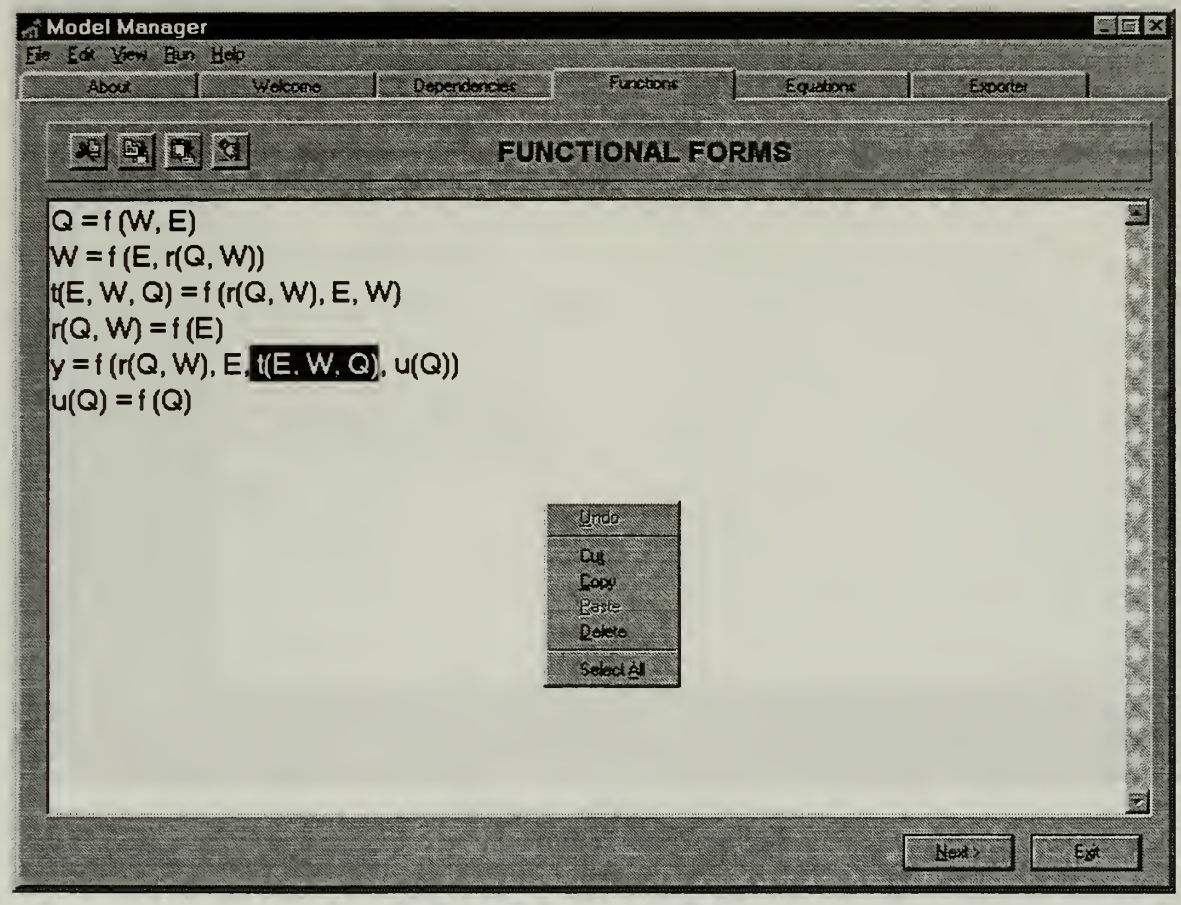


Figure 20. Functions Page

Once the modeler believes that the functional forms are what he/she wanted, then he/she must press the "Next >" button to create the real equations. These results, which appear in "Equations" page, are generated by using the modeling rules (indexical equivalence, dimensional balance, and model structure). Then the user should refine and validate the equations, which are suggested by the program. After the changes and corrections are made, the program checks the syntax and logic errors in order to verify the validity of the equations again. Errors in areas such as parenthesis and punctuation can virtually be eliminated. The equations page is almost the same as the dependencies page. It is also an editor with the same functionalities. Figure 21 shows the equations page.

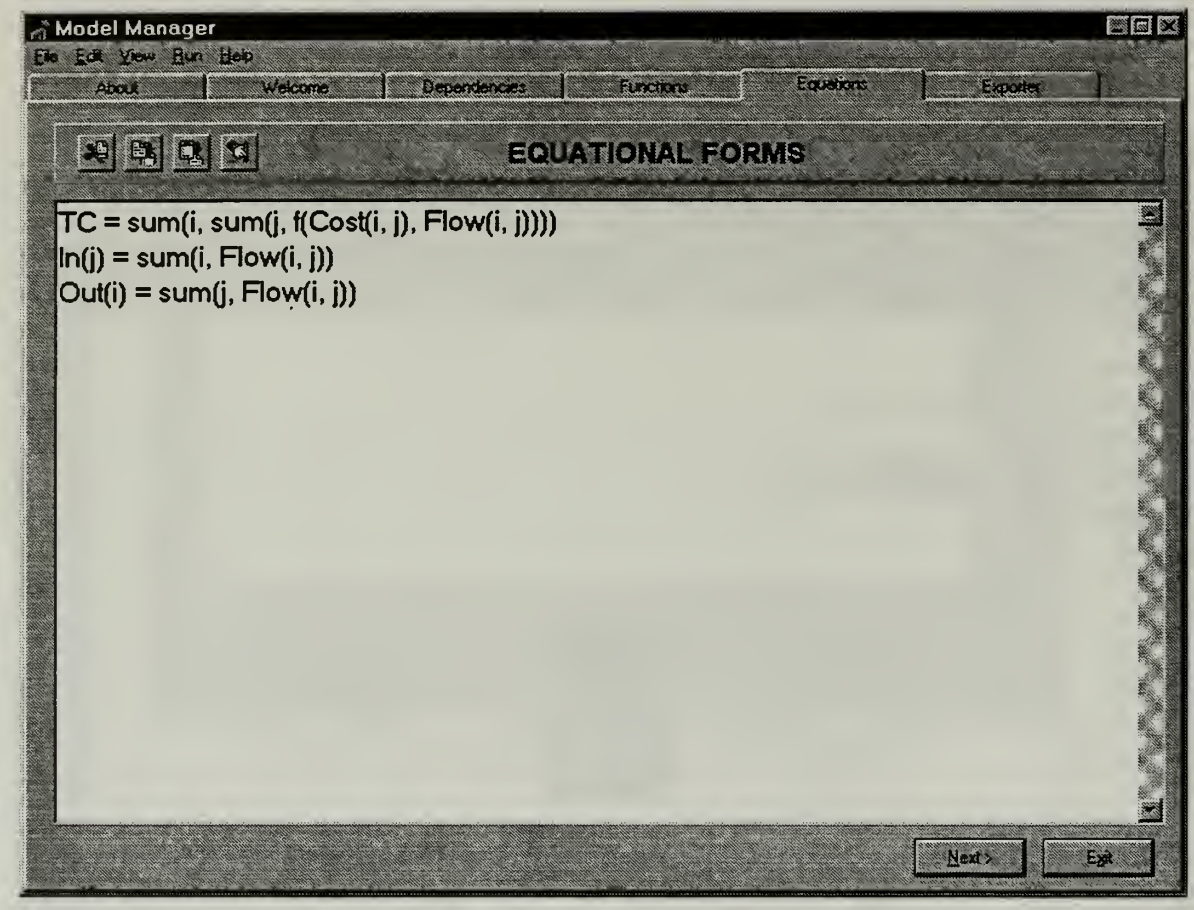


Figure 21. Equations Page

Finally, when the modeler presses the "Next >" button in the equations page, the program launches the "Exporter" page, where the actual code is generated by the computer. In the exporter page, Figure 22, there are three options to choose from. They are :

AMPL, GAMS, and HTML. When the modeler selects any one of them, then the program generates the actual code for that language. The user can save as a file format of the modeling language of choice or HTML document.

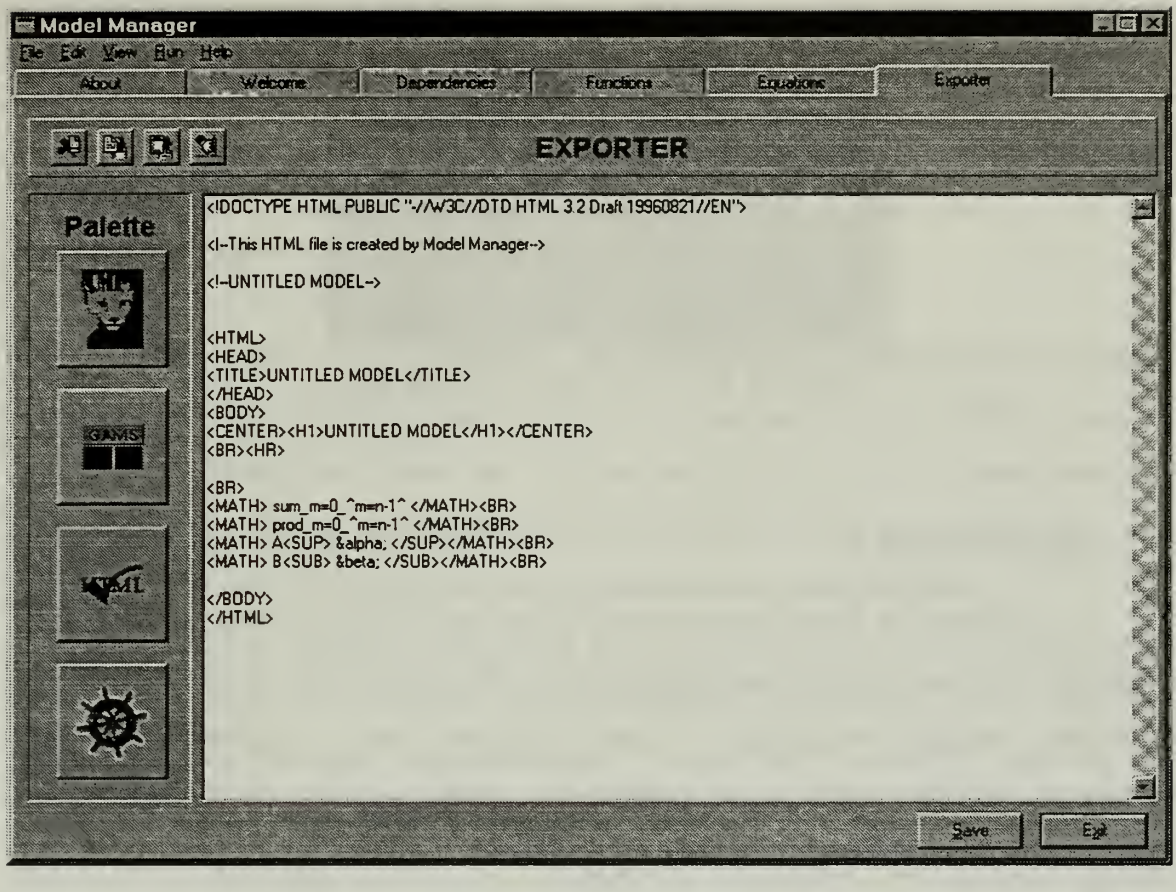


Figure 22. Exporter Page

If the modeler selects the HTML button, then he can also browse this code by running the Netscape Navigator directly from the program. He/she should press the following button in order to that:

The user can exit from the program anytime by clicking the exit button or the close icon of the title bar:

When the user wants to close the program a message dialog box, shown in Figure 23, appears to confirm the command.

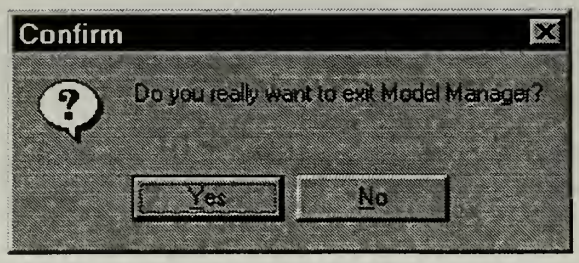


Figure 23. Exit Confirmation Dialog Box

C. MAINTENANCE

In looking at the Visual Development Methodology (VDM), it becomes clear that the concept of system development phases becomes very blurred. The difference between requirements analysis and design, or the difference between implementation and maintenance, becomes harder to see because the building of the applications pervades virtually every area of the entire development cycle.

Development begins in the very early stages of investigating the problem to be solved. Solving the problem identified is not the completion of the development process. Instead, this process will continue indefinitely. As long as there is a need for the application, developers should continue working on it. What was maintenance becomes implementation.

The VDM means a redefining of the meaning of the maintenance phase. The VDM does not draw a line between maintenance and development. Maintenance becomes a part of the original development process itself and begins very early in the development effort. Therefore, the project presented in this thesis should be viewed as being in the maintenance mode right from the beginning, rather than at the end of the development life cycle.

Throughout the development process, Delphi allowed the developer to make changes, often right in the presence of the users, to ensure that the interface design was satisfactory and met the needs and desires of those who will use the system.

V. CONCLUSIONS

A. SUMMARY

In this thesis, it is suggested that diagrammatic representations improve comprehension and simplify problem structuring. In support of this, the thesis proposes dependency diagrams as a tool for problem structuring and as a foundation for development of a formal mathematical model. It proposes that dependencies between variables can be used for building and using models and modeling languages.

An important aspect of this thesis is the development of a GUI-based model formulation interface for algebraic modeling languages and systems. These interfaces facilitate the description and implementation of mathematical models by allowing the modeler to use universal mathematical symbols and algebraic notation. This approach enhances the development process by replacing the requirements of language specific codes with a generalized modeling environment, which provides better visualization of model components and more interactive model management.

This thesis also shows that dependency graphs can play a useful role in model management. These dependencies support almost every step of a modeling life cycle. They serve not only the model formulation phase by representing the qualitative problem in a more structured way, but also serve as model documentation, which should facilitate model maintenance and model reuse. Automatic validation can be performed. The documentation is active; that makes it more likely to be used. Further, it does not just act as a guard; it actually can help in the development of the model specification. That is, the "code" that represents the model can partially be generated based on the dependency graph. So, this graphical representation is an integrated approach towards model management.

B. CONTRIBUTIONS

One of the contributions of this thesis is that it highlights the lack of literature in providing guidance on *computational dependencies between model components* and *the semantics of indexing operations and expressions in a clear, implementation-independent manner*. In other domains, there is significant work on this topic which make use of these dependencies and their relations extensively. Dependency-based reasoning is more established in software engineering and software design than in mathematical modeling.

Another contribution is the observation that although most current algebraic modeling languages are run on top of GUI-based operating systems, they are still file-oriented and text-based. Therefore, these languages require structured declarations and formal model definitions. They use strict syntax rules, notations, code words and data entry methods. Rosenthal [Ref. 17: pp. 7-32] explains the importance of the correct implementation of these rules with many informative bullets in his GAMS tutorial. Since average users generally cannot remember/control all of these rules, the use of the system becomes more error-prone. Whereas, in a GUI environment, most of the errors mentioned here would be eliminated automatically because the utilization of graphical objects forces the users to enter their inputs in a *"standard"* way. If the primary purpose and demand of GUI computing is to make computers easier to use for everyone, then why should this demand not be applied to the current modeling languages? This thesis has provided some insight that modelers can improve usability features and ease-of-use by adding an intermediate interface between the user and the modeling languages of today. This approach not only enables modelers to present the whole modeling process clearly but also helps to "*see the big picture*" in model management.

The most significant contribution this thesis has made is to provide a tool, *dependency diagrams*, for problem structuring and as a foundation for the development of a formal mathematical model. The incorporation of this tool into modeling languages and systems facilitates model formulation, validation, maintenance and reuse.

C. AREAS OF FURTHER RESEARCH

The rapid growth of the Internet in general, the explosive growth of the World Wide Web in particular, creates new opportunities for the development and deployment of decision technologies for and by organizations and individuals [REF. 30]. The application presented in this thesis is well-suited for web-based development that would make it accessible to a broad range of users. By using Java technology, portability and platform-independence issues can be resolved. The user interface can also be improved.

If it is implemented in the Java programming language, Model Manager could also be registered as a provider on *DecisionNet* [REF. 31, 32, 33, 34, 35, 36, and 37]. DecisionNet is a distributed, Web-based electronic market for decision technologies such as data, models, solution algorithms, and modeling environments. It acts as a broker between "providers" (owners of decision support technologies) and "consumers" (users of decision support technologies). This web-based approach also creates an opportunity for modelers to build their own models on the WWW. After building the model in Model Manager, the output file can be sent to the solvers through DecisionNet. DecisionNet allows a consumer to use a decision technology when he/she wants it. Also the consumer is free of the problem of owning and managing the software (use vs. own).

If dependencies between model variables, as represented in dependency diagrams, are incorporated into modeling languages and systems in a GUI environment, they could be more useful in model formulation and the other phases of the modeling life-cycle.

We hope that further enhancements will be made to this system, increasing its capability, web-orientation and ease-of-use.

LIST OF REFERENCES

1. *The American Heritage® Dictionary of the English Language,* Third Edition, Houghton Mifflin Company, 1992.

2. Taha, A. H., *Operations Research: An Introduction*, 5th Edition, MacMillan Publishing Company, Engelwood Cliffs, New Jersey, 1992.

3. Dolk, D. R., "An Introduction to Model Integration and Integrated Modeling Environments," *Decision Support Systems*, 10, pp.249-254, 1993.

4. Dolk, D. R., "Model Management and Structured Modeling: The Role of an Information Resource Dictionary System," *ACM* 31, 6, pp.704-718, 1988.

5. Sprague, Ralph, "A Framework for the Development of Decision Support Systems," *MIS Quarterly*, 4(2), pp. 1-26, 1980.

6. Sprague, R.H. and Carlson, E. D., *Building Effective Decision Support Systems*, Prentice Hall, Englewood Cliffs, New Jersey, 1982.

7. Bhargava, H. K. and Kimbrough S.O., "Model Management: An Embedded Languages Approach," *Decision Support Systems,* 10:3, pp. 277-300, 1993.

8. Liang, T.P., "Integrating Model Management with Data Management in Decision Support Systems," *Decision Support Systems*, 1, pp. 221-232, 1985.

9. Gagliardi, M. and Spera, C. "MODASS: An Object Oriented Model Management System," *Proceedings of the INFORMS Conference on Information Systems,* Washington, DC, 1995.

10. Saaty, Thomas L. and Alexander, J. M., *Thinking With Models: Mathematical Models in the Physical, Biological, and Social Sciences*, Pergamon Press, New York, New York, 1981.

11. Bhargava, H. K., *Lecture Notes for Decision Support Systems*, Naval Postgraduate School, Monterey, California, 1996.

12. Chari, K and Sen, T. K., "An Implementation of a Graph Based Modeling System for Structured Modeling (GBMS/SM)," *Working Paper*, James Madison University, Harrisonburg, Virginia, 1997.

13. Geoffrion, A. M., "Structured Modeling: Survey and Future Research Directions," *ORSA CSTS Newsletter,* 15: 1, 1994. (Updated version, May 1996 at Web site: http://personal.anderson.ucla.edu/~art.geoffrion/home/docs/csts/index.htm.)

14. Geoffrion, A.M., "An Introduction to Structured Modeling," Management Science, 33:5, pp. 547-588, 1987. WMSI Reprint 219. Formerly WMSI Working Paper 338, 6/86, (revised 12/86, 2/87, 5/87, 8/87, 3/88).

15. Fourer, Robert, "Modeling Languages versus Matrix Generators for Linear Programming," *ACM Transactions on Mathematical Software*, 9, pp.143-183, 1983.

16. Fourer, Robert, Gay, David M., and Kernighan, Brian W., *AMPL A Modeling Language for Mathematical Programming,* p.xiii, Boyd & Fraser Publishing Company, Danvers, Massachusetts, 1993.

17. Brooke, A., Kendrick, D., and Meeraus, A., *GAMS, Release 2.25, A User's Guide*, Boyd & Fraser Publishing Company, Danvers, Massachusetts, 1992.

18. Blanning, R.W, *Model Management Systems: An Overview*, Owen Graduate School of Management, Vanderbilt University, Nashville, Working Paper No. 89-23, 1989.

19. Baldwin, A. A., Baldwin, D., and Sen, T., "The Evolution and Problems of Model Management Research," *Omega*, 19(6), pp. 511-528, 1991.

20. Liang, T. P. and Jones, C. V., "Meta-Design Considerations in Developing Model Management Systems," *Decision Sciences*, 19(1), pp. 72-92, 1988.

21. Pracht, W. E., "Model Visualization: Graphical Support for DSS Problem Structuring and Knowledge Organization," *Decision Support Systems*, 6, pp. 13-27, 1990.

22. Sen, T., "Diagrammatic Knowledge Representation," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 22(4), pp. 826-830, 1992.

23. Larkin, J. H. and Simon, H. A., "Why a Diagram is (Sometimes) Worth Ten Thousand Words," *Cognitive Science*, 11, pp. 65-99, 1987.

24. Bhargava, H. K., "Model Management via Dependencies Between Variables," *Working Paper*, Department of Systems Management, Naval Postgraduate School, Monterey, California, 1996.

25. Bhargava, H. K., "Indexical Reasoning in Mathematical Modeling," *Working Paper*, Department of Systems Management, Naval Postgraduate School, Monterey, California, 1996.

26. Bhargava, H. K., "Dimensional Analysis in Mathematical Modeling Systems: A Simple Numerical Method," *ORSA Journal on Computing*, 5: 1, pp. 33-39, 1993.

27. Bradley, G. H. and Clemence Jr., R. D., "Model Integration with a Typed Executable Modeling Language," *Proceedings of the 21st Annual Hawaii International Conference on System Sciences,* Vol. III, IEEE Computer Society Press, Los Alamitos, California, pp. 403-410, 1988.

28. Bradley, G. H. and Clemence Jr., R. D., "A Type Calculus for Executable Modeling Languages," *IMA Journal of Mathematics in Management*, 1, pp. 277-291, 1987.

29. Borland International, Inc., *Borland Delphi™ Database Application Developer's Guide*, Scotts Valley, California, 1995.

30. Bhargava, H.K., Krishnan, R., and Muller, R.,"Electronic Markets for Decision Technologies: A Business Cycle Analyis," forthcoming in *International Journal of Electronic Commerce*, Last revised October 1996.

31. Bhargava, H.K., Krishnan, R., and Muller, R., "Decision Support on Demand: On Emerging Electronic Markets for Decision Technologies," forthcoming in *Decision Support Systems*, Last revised May 1996.

32. Bhargava, H.K., Krishnan, R., Roehrig, S., Kaplan, D., Casey, M. P., and Muller, R., "Model Management in Electronic Markets for Decision Technologies: A Software Agent Approach," *Proceedings of the Thirtieth Hawaii International Conference on System Sciences*, (Maui, Hawai'i), January 1997.

33. Bhargava, H.K., Krishnan, R., and Muller, R., "On Parameterized Transaction Models for Agents in Electronic Markets for Decision Technologies," *Proceedings of the Fifth International Workshop on Information Technologies*, Amsterdam, December 1995.

34. Bhargava, H.K., Krishnan, R., and Muller, R., "On Sharing Decision Technologies over a Global Network," *Proceedings of the International Conference on Automation*, (Indore, India), December 1995.

35. Bhargava, H.K., Krishnan, R., and Kaplan, D., "On Generalized Access to a WWW-based Network of Decision Support Services," *Proceedings of the Third ISDSS Conference*, Hong Kong, June 1995.

36. Bhargava, H. K., King, A. S., and McQuay, D. S., "DecisonNet: An Architecture for Modeling and Decision Support over the World Wide Web," *Proceedings of the Third International Conference on Decision Support Systems*, Vol. 2, 1995.

37. Bhargava, H. K., DecisonNet Web site, http://dnet.sm.nps.navy.mil/.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center 8725 John J. Kingman Road., Ste 0944 Ft. Belvoir, VA 22060-6218	2
2.	Dudley Knox Library Naval Postgraduate School 411 Dyer Rd. Monterey, California 93943-5101	2
3.	Professor Hemant K. Bhargava (Code SM/BH) Department of Systems Management Naval Postgraduate School Monterey, California 93943-5000	2
4.	Professor Gordon H. Bradley (Code OR/BZ) Department of Operations Research Naval Postgraduate School Monterey, California 93943-5000	1
5.	Professor Daniel Dolk (Code SM/DK) Department of Systems Management Naval Postgraduate School Monterey, California 93943-5000	1
6.	Professor Bala Ramesh (Code SM/RA) Department of Systems Management Naval Postgraduate School Monterey, California 93943-5000	1
7.	Devrim Rehber Plevne Mah. Pilot Sok. Arik Apt. No: 28/1 10050 Balikesir TURKEY	2

8. Dz. K. K.
Personel Egitim Daire Baskanligi 1
06100 Bakanliklar / ANKARA
TURKEY

9. Deniz Harp Okulu Komutanligi 1
Ogretim Baskanligi
Tuzla/Istanbul
TURKEY

10. Bogazici University 1
80815 Bebek/Istanbul
TURKEY

11. ODTU 1
06531 Ankara
TURKEY

12. Bilkent University 1
Department of Computer Engineering
and Information Science
06533 Bilkent/Ankara
TURKEY

13. Professor Arthur M. Geoffrion 1
Decision Sciences
John E. Anderson Graduate School of Management
Box 951481, UCLA
Los Angeles, CA 90095-1481

14. Professor Tarun K. Sen 1
Department of Accounting
Pamplin Collage of Business
Virginia Polytechnic Institude and State University
Blacksburg, VA 24061-0101

15. Professor Ramayya Krishnan — 1
Decision Systems Research Institute
The Heinz School
Carnegie Mellon University
Pittsburgh, PA 15213

16. Professor Chris Jones — 1
Department of Management Science
School of Business Administration
University of Washington
Seattle, WA 98195

17. Capt. Dan S. McQuay, USMC — 1
1 Westwood Court
Stafford, VA 22135

18. Sanjay Saigal — 1
Compass Modeling Solutions, Inc.
1005 Terminal Way, Suite 100
Reno, NV 89502

19. Professor Robert Fourer — 1
Dept. of Industrial Engineering and Management Sciences
Northwestern University
2225 North Campus Drive
Evanston, IL 60208-3119

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

DUDLEY KNOX LIBRARY
3 2768 00336069 4